ursgal Documentation

Release 0.6.8

**Lukas P. M. Kremer,
Purevdulam Oyunchimeg,
Johannes Barth,
Stefan Schulze and
Christian Fufezan**

Feb 19, 2021

Contents

The latest Documentation was generated on: Feb 19, 2021

CHAPTER 1

Introduction

Ursgal - Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis

1.1 Summary

Ursgal is a Python module that offers a generalized interface to common bottom-up proteomics tools, e.g.

a) Peptide spectrum matching with up to eight different search engines (some available in multiple versions), including four open modification search engines

b) Evaluation and post processing of search results with up to two different engines for protein database searches as well as two engines for the post processing of mass difference results from open modification engines

c) Integration of search results from different search engines

d) De novo sequencing with up to four different search engines

e) Miscellaneous tools including the creation of a target decoy database as well as filtering, sanitizing and visualizing of results

1.2 Abstract

Proteomics data integration has become a broad field with a variety of programs offering innovative algorithms to analyze increasing amounts of data. Unfortunately, this software diversity leads to many problems as soon as the data is analyzed using more than one algorithm for the same task. Although it was shown that the combination of multiple peptide identification algorithms yields more robust results (Nahnsen et al. 2011, Vaudel et al. 2015, Kwon et al. 2011), it is only recently that unified approaches are emerging (Vaudel et al. 2011, Wen et al. 2015); however, workflows that, for example, aim to optimize search parameters or that employ cascaded style searches (Kertesz-Farkas et al. 2015) can only be made accessible if data analysis becomes not only unified but also and most importantly scriptable. Here we introduce Ursgal, a Python interface to many commonly used bottom-up proteomics tools and to additional auxiliary programs. Complex workflows can thus be composed using the Python scripting

language using a few lines of code. Ursgal is easily extensible, and we have made several database search engines (X!Tandem (Craig and Beavis 2004), OMSSA (Geer et al. 2004), MS-GF+ (Kim et al. 2010), Myrimatch (Tabb et al. 2008), MS Amanda (Dorfer et al. 2014)), statistical postprocessing algorithms (qvality (Käll et al. 2009), Percolator (Käll et al. 2008)), and one algorithm that combines statistically postprocessed outputs from multiple search engines ("combined FDR" (Jones et al. 2009)) accessible as an interface in Python. Furthermore, we have implemented a new algorithm ("combined PEP") that combines multiple search engines employing elements of "combined FDR" (Jones et al. 2009), PeptideShaker (Vaudel et al. 2015), and Bayes' theorem.

Kremer, L. P. M., Leufken, J., Oyunchimeg, P., Schulze, S. and Fufezan, C. (2015): Ursgal, Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis , Journal of Proteome research, 15, 788-. DOI:10.1021/acs.jproteome.5b00860

1.3 Documentation

The complete Documentation can be found at Read the Docs

Besides the Download and Installation steps, this includes a Quick Start Tutorial detailed documentation of the Modules and Available Engines as well as a broad set of Example Scripts and many more.

1.4 Download and Installation

Ursgal requires Python 3.4 or higher. If you want to run Ursgal on a Windows system, Python 3.6 or higher is recommended.

There are two recommended ways for installing Ursgal:

- Installation via pip
- Installation from the source (GitHub)

1.4.1 Installation via pip

Execute the following command from your command line:

```
user@localhost:~$ pip install ursgal
```

This installs Python into your Python site-packages.

To download the executables, which we are allowed to distribute run:

```
user@localhost:~$ ursgal-install-resources
```

You can now use it with all engines that we have built or that we are allowed to distribute. For all other third-party engines, a manual download from the respective homepage is required (see also: How to install third party engines)

Note: Pip is included in Python 3.4 and higher. However, it might not be included in in your system's PATH environment variable. If this is the case, you can either add the Python scripts directory to your PATH env variable or use the path to the pip.exe directly for the installation, e.g.: ~/Python34/Scripts/pip.exe install ursgal

Note: On Mac it may be neccesary to use Python3.6, since it comes with its own OpenSSL now. This may avoid problems when using pip.

1.4.2 Installation from source

1. Download Ursgal using GitHub **or** the zip file:

- GitHub version: Starting from your command line, the easiest way is to clone the GitHub repo.:

```
user@localhost:~$ git clone https://github.com/ursgal/ursgal.git
```

- ZIP version: Alternatively, download and extract the ursgal zip file

2. Next, navigate into the Ursgal folder and install the requirements:

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

Note: Pip is included in Python 3.4 and higher. However, it might not be included in in your system's PATH environment variable. If this is the case, you can either add the Python scripts directory to your PATH env variable or use the path to the pip.exe directly for the installation, e.g.: ~/Python34/Scripts/pip.exe install -r requirements.txt

---

Note: On Mac it may be neccesary to use Python3.6, since it comes with its own OpenSSL now. This may avoid problems when using pip.

---

3. Finally, use setup.py to download third-party engines (those that we are allowed to distribute) and to install Ursgal into the Python site-packages:

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

If you want to install the third-party engines without installing Ursgal into the Python site-packages you can use:

```
user@localhost:~/ursgal$ python setup.py install_resources
```

Note: Since we are not allowed to distribute all third party engines, you might need to download and install them on your own. See FAQ (How to install third party engines) and the respective engine documentation for more information.

Note: Under Linux, it may be required to change the permission in the python site-package folder so that all files are executable

---

(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.)

1.5 Tests

Run tox in root folder. You might need to install tox for Python3 first although it is in the requirements_dev.txt (above) thus pip install -r requirements_dev.txt should have installed it already. Then just execute:

```
user@localhost:~/ursgal$ tox
```

In case you only want to test one python version (e.g because you only have one installed), run for e.g. python3.5:

```
user@localhost:~/ursgal$ tox -e py35
```

For other environments to run, check out the tox.ini file to test the package.

1.6 Update to v0.6.0 Warning

Please note that, due to significant reorganization of UController functions as well as some uparams, compatibility of v0.6.0 with previous versions is not given in all cases. Most likely, your previous results will not be recognized, i.e. previously executed runs will be executed again. Please consider this before updating to v0.6.0, check the Changelog or ask us if you have any doubts. We are sorry for the inconvenience but changes were necessary for further development. If you want to continue using (and modifying) v0.5.0 you can use the branch v0.5.0.

1.7 Questions and Participation

If you encounter any problems you can open up issues at GitHub, join the conversation at Gitter, or write an email to ursgal.team@gmail.com. Please also check the Frequently Asked Questions.

For any contributions, fork us at https://github.com/ursgal/ursgal and open up pull requests! Please also check the *Contribution Guidelines*. Thanks!

1.8 Disclaimer

Ursgal is beta and thus still contains bugs. Verify your results manually and as common practice in science, never trust a blackbox :)

1.9 Copyrights

1.10 Contact

Dr. Christian Fufezan
Institute of Pharmacy and Molecular Biotechnology
Heidelberg University
Germany
eMail: christian@fufezan.net

1.11 Citation

In an academic world, citations are the only credit that one can hope for ;) Therefore, please do not forget to cite us if you use Ursgal:

Schulze, S., Igiraneza, A. B., Kösters, M., Leufken, J., Leidel, S. A., Garcia, B. A., Fufezan, C., and Pohlschroder, M. (2021) Enhancing Open Modification Searches via a Combined Approach Facilitated by Ursgal Journal of Proteome Research, DOI:10.1021/acs.jproteome.0c00799

Kremer, L. P. M., Leufken, J., Oyunchimeg, P., Schulze, S., and Fufezan, C. (2016) Ursgal, Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis Journal of Proteome Research 15, 788–794, DOI:10.1021/acs.jproteome.5b00860

Note: Please also cite every tool you use in Ursgal. During runtime the references of the tools you are using are shown.

Full list of tools with proper citations that are integrated into Ursgal are:

- Craig, R.; Beavis, R. C. TANDEM: matching proteins with tandem mass spectra. Bioinformatics 2004, 20 (9), 1466–1467.
- Dorfer, V.; Pichler, P.; Stranzl, T.; Stadlmann, J.; Taus, T.; Winkler, S.; Mechtler, K. MS Amanda, a Universal Identification Algorithm Optimised for High Accuracy Tandem Mass Spectra. J. Proteome Res. 2014.
- Frank, A. M.; Savitski, M. M.; Nielsen, M. L.; Zubarev, R. A. and Pevzner, P. A. De Novo Peptide Sequencing and Identification with Precision Mass Spectrometry. J. Proteome Res. 2007 6:114-123.',
- Geer, L. Y.; Markey, S. P.; Kowalak, J. A.; Wagner, L.; Xu, M.; Maynard, D. M.; Yang, X.; Shi, W.; Bryant, S. H. Open Mass Spectrometry Search Algorithm. J. Proteome res. 2004, 3 (5), 958–964.
- Hoopmann, M. R.; Zelter, A.; Johnson, R. S.; Riffle, M.; Maccoss, M. J.; Davis, T. N.; Moritz, R. L. Kojak: Efficient analysis of chemically cross-linked protein complexes. J Proteome Res 2015, 14, 2190-198
- Jones, A. R.; Siepen, J. a.; Hubbard, S. J.; Paton, N. W. Improving sensitivity in proteome studies by analysis of false discovery rates for multiple search engines. Proteomics 2009, 9 (5), 1220–1229.
- Kim, S.; Mischerikow, N.; Bandeira, N.; Navarro, J. D.; Wich, L.; Mohammed, S.; Heck, A. J. R.; Pevzner, P. A. The generating function of CID, ETD, and CID/ETD pairs of tandem mass spectra: applications to database search. MCP 2010, 2840–2852.
- Käll, L.; Canterbury, J. D.; Weston, J.; Noble, W. S.; MacCoss, M. J. Semi-supervised learning for peptide identification from shotgun proteomics datasets. Nature methods 2007, 4 (11), 923–925.
- Käll, L.; Storey, J. D.; Noble, W. S. Qvality: Non-parametric estimation of q-values and posterior error probabilities. Bioinformatics 2009, 25 (7), 964–966.

- Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature methods 2017, 14, 513–520
- Leufken J, Niehues A, Sarin LP, Wessel F, Hippler M, Leidel SA, Fufezan C. pyQms enables universal and accurate quantification of mass spectrometry data. Mol Cell Proteomics 2017, 16, 1736-1745
- Ma, B. Novor: real-time peptide de novo sequencing software. J Am Soc Mass Spectrom. 2015 Nov;26(11):1885-94
- Na S, Bandeira N, Paek E. Fast multi-blind modification search through tandem mass spectrometry. Mol Cell Proteomics 2012, 11
- Reisinger, F.; Krishna, R.; Ghali, F.; Ríos, D.; Hermjakob, H.; Antonio Vizcaíno, J.; Jones, A. R. JmzIdentML API: A Java interface to the mzIdentML standard for peptide and protein identification data. Proteomics 2012, 12 (6), 790–794.
- Tabb, D. L.; Fernando, C. G.; Chambers, M. C. MyriMatch: highly accurate tandem mass spectral peptide identification by multivariate hypergeometric analysis. J Proteome Res. 2008, 6 (2), 654–661.
- Yu, F., Li, N., Yu, W. PIPI: PTM-Invariant Peptide Identification Using Coding Method. J Prot Res 2016, 15
- Barsnes, H., Vaudel, M., Colaert, N., Helsens, K., Sickmann, A., Berven, F. S., and Martens, L. (2011) compomics-utilities: an open-source Java library for computational proteomics. BMC Bioinformatics 12, 70
- Leufken, J., Niehues, A., Sarin, L. P., Wessel, F., Hippler, M., Leidel, S. A., and Fufezan, C. (2017) pyQms enables universal and accurate quantification of mass spectrometry data. Mol. Cell. Proteomics 16, 1736–1745
- Jaeger, D., Barth, J., Niehues, A., and Fufezan, C. (2014) pyGCluster, a novel hierarchical clustering approach. Bioinformatics 30, 896–898
- Bald, T., Barth, J., Niehues, A., Specht, M., Hippler, M., and Fufezan, C. (2012) pymzML–Python module for high-throughput bioinformatics on mass spectrometry data. Bioinformatics 28, 1052–1053
- Kösters, M., Leufken, J., Schulze, S., Sugimoto, K., Klein, J., Zahedi, R. P., Hippler, M., Leidel, S. A., and Fufezan, C. (2018) pymzML v2.0: introducing a highly compressed and seekable gzip format. Bioinformatics 34, 2513-2514
- Liu, M.Q.; Zeng, W.F.; Fang, P.; Cao, W.Q.; Liu, C.; Yan, G.Q.; Zhang, Y.; Peng, C.; Wu, J.Q.;
- Zhang, X.J.; Tu, H.J.; Chi, H.; Sun, R.X.; Cao, Y.; Dong, M.Q.; Jiang, B.Y.; Huang, J.M.; Shen, H.L.; Wong ,C.C.L.; He, S.M.; Yang, P.Y. (2017) pGlyco 2.0 enables precision N-glycoproteomics with comprehensive quality control and one-step mass spectrometry for intact glycopeptide identification. Nat Commun 8(1)
- Yuan, Z.F.; Liu, C.; Wang, H.P.; Sun, R.X.; Fu, Y.; Zhang, J.F.; Wang, L.H.; Chi, H.; Li, Y.; Xiu, L.Y.; Wang, W.P.; He, S.M. (2012) pParse: a method for accurate determination of monoisotopic peaks in high-resolution mass spectra. Proteomics 12(2)
- Hulstaert, N.; Sachsenberg, T.; Walzer, M.; Barsnes, H.; Martens, L. and Perez-Riverol, Y. (2019) ThermoRawFileParser: modular, scalable and cross-platform RAW file conversion. bioRxiv https://doi.org/10.1101/622852
- Tran, N.H.; Zhang, X.; Xin, L.; Shan, B.; Li, M. (2017) De novo peptide sequencing by deep learning. PNAS 114 (31)
- Devabhaktuni, A.; Lin, S.; Zhang, L.; Swaminathan, K.; Gonzalez, CG.; Olsson, N.; Pearlman, SM.; Rawson, K.; Elias, JE. (2019) TagGraph reveals vast protein modification landscapes from large tandem mass spectrometry datasets. Nat Biotechnol. 37(4)
- Yang, H; Chi, H; Zhou, W; Zeng, WF; He, K; Liu, C; Sun, RX; He, SM. (2017) Open-pNovo: De Novo Peptide Sequencing with Thousands of Protein Modifications. J Proteome Res. 16(2)
- Polasky, DA; Yu, F; Teo, GC; Nesvizhskii, AI (2020) Fast and comprehensive N- and O-glycoproteomics analysis with MSFragger-Glyco. Nat Methods 17 (11)

- Geiszler, DJ; Kong, AT; Avtonomov, DM; Yu, F; Leprevost, FV; Nesvizhskii, AI (2020) PTM-Shepherd: analysis and summarization of post-translational and chemical modifications from open search results. bioRxiv doi: https://doi.org/10.1101/2020.07.08.192583
- An, Z; Zhai, L; Ying, W; Qian, X; Gong, F; Tan, M; Fu, Y. (2019) PTMiner: Localization and Quality Control of Protein Modifications Detected in an Open Search and Its Application to Comprehensive Post-translational Modification Characterization in Human Proteome. Mol Cell Proteomics 18 (2)
- 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

Quick Start

2.1 Quick Start Tutorial

This tutorial will explain the basic usage of Ursgal using simple examples. To get started, make sure you have *installed Ursgal*.

2.1.1 1. Getting Started

Once you installed Ursgal, you should be able to import it in your Python3 scripts like this:

```
import ursgal
```

To get an overview over the engines that are available on your computer, initialize the Ursgal UController class. This should print a list of engines to your screen, sorted by category:

```
uc = ursgal.UController()
```

The UController controls, manages and executes the tools that are available in Ursgal. These tools are called UNodes. Some UNodes (especially search engines) require binary executable files, which are not included in Ursgal by default. If the UController overview shows a lot of missing search engines, you probably have not executed the script 'install_resources.py' in the Ursgal directory (see: *Installation*). This script automatically downloads third-party tools. 'install_resources.py' should be executed before 'setup.py'. If you did it the other way around, you have to re-run 'setup.py' once again.

2.1.2 2. Running a Simple Search

One of the key features of Ursgal is peptide spectrum matching with up to five search engines. To perform a search, you need:

- a peptide database (.fasta) that will be searched
- one or more mass spectrometer output files (.mzML or .mgf)

Once you have these files, you are ready to execute a full search with Ursgal:

```
import ursgal
uc = ursgal.UController(
    params = {'database': 'my_database.fasta'}
)

search_result = uc.search(
    input_file = 'my_mass_spec_file.mzML',
    engine     = 'omssa',
)
```

This will produce a .csv file containing the peptide-spectrum-matches (PSMs) found by the specified search engine. The above example uses the search engine OMSSA. To use a different search engine, simply replace the engine keyword argument 'omssa' of *`UController.search()`* with the name of a different engine (see: *Available Engines*).

Note: Several engines have their own requirements, e.g. Java based engines like MS-GF+ or MSFragger require the 'Java Runtime Environment <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>'. Please make sure, all required software is installed.

2.1.3 3. Adjusting Parameters

If you used OMSSA or any other search engine before, you will know that there are a lot of search parameters and settings that can be defined. For instance, depending on the mass spectrometer that was used, you might want to set the fragment mass tolerance unit to Dalton or ppm. In Ursgal, there are two ways to adjust such parameters:

```
# 1) define parameters at UController initialization:
uc = ursgal.UController(
    params = {
        'database': 'my_database.fasta',
        'frag_mass_tolerance':      0.5,
        'frag_mass_tolerance_unit': 'da',
    }
)

# 2) change parameters after UController is already initialized:
uc.params['database'] = 'my_other_database.fasta'
uc.params['frag_mass_tolerance']      = 15
uc.params['frag_mass_tolerance_unit'] = 'ppm'
```

The second method allows you to re-adjust parameters at different points of your Python script.

For a list of available ursgal parameters, see *Parameters*. Ursgal also includes pre-defined sets of parameters for different mass spectrometers. These are called profiles. Currently, three profiles are available: 'LTQ XL low res', 'LTQ XL high res' and 'QExactive+'. Profiles can be used like this:

```
uc = ursgal.UController(
    params  = {'database': 'my_database.fasta'},
    profile = 'QExactive+'
)
```

2.1.4 4. Available Workflow Functions

You have already seen the *UController.search()* function in section 2. *UController.search()* is only one of many UController functions that you can use to define custom workflows. A commonly used procedure is to post-process search engine results with tools such as Percolator or qvality. These tools can be accessed using the *UController.validate()* function. In this example, we use Percolator to discriminate correct from incorrect peptide-spectrum matches and calculate posterior error probabilities:

```
search_result = uc.search(
    input_file = 'my_mass_spec_file.mzML',
    engine     = 'omssa',
)

validated_result = uc.validate(
    input_file = search_result,
    engine     = 'percolator_2_08',
)
```

Currently, the following UController workflow functions are available:

- ***UController.combine_search_results()*** Statistical integration of search results from multiple engines
- ***UController.convert()*** Conversion from and into different file formats (*Converter Engines*)
- ***UController.execute_misc_engine()*** Can be used to execute any available engine (*Misc Engines*)
- ***UController.fetch_file()*** Downloads files (HTTP or FTP)
- ***UController.quantify()*** Quantification of identified molecules.
- ***UController.search()*** Peptide spectrum matching
- ***UController.validate()*** Statistical post-processing of search results to calculate the false-discovery rate (FDR, q-value) and/or posterior error probability (PEP) (*Validation Engines*)
- ***UController.visualize()*** Visualization of results (*Visualizer*)

2.1.5 5. Building Custom workflows

The above functions can be used in conjunction with standard Python control flow tools such as loops and if-statements. This makes it possible to define complex and highly customizable workflows. For instance, imagine you have multiple mzML files and you want to use all available search engines with them:

```
spec_files     = ['fileA.mzML', 'fileB.mzML']
search_engines = ['omssa_2_1_9', 'xtandem_alanine', 'msgfplus_v2017_01_27',
                  'msamanda_2_0_0_9695', 'myrimatch_2_1_138']
```

This task can be easily achieved with the power of nested for-loops:

```
results = []
for spec_file in spec_files:
    for search_engine in search_engines:
        result = uc.search(
            input_file = spec_file,
            engine     = search_engine,
        )
        results.append( result )
```

The above script will generate ten output files (search results of two mzML files per engine):

```
>>> print( results )
['fileA_omssa_2_1_9_unified.csv', 'fileB_omssa_2_1_9_unified.csv',
 'fileA_xtandem_alanine_unified.csv', 'fileB_xtandem_alanine_unified.csv',
 'fileA_msgfplus_v2017_01_27_unified.csv', 'fileB_msgfplus_v2017_01_27_unified.csv',
 'fileA_msamanda_2_0_0_9695_unified.csv', 'fileB_msamanda_2_0_0_9695_unified.csv',
 'fileA_myrimatch_2_1_138_unified.csv', 'fileB_myrimatch_2_1_138_unified.csv']
```

2.1.6 6. JSONs, Force and File Names

If you execute the same Ursgal script twice, you will notice that the UNodes are not executed in the second run. This is because Ursgal notes down the input file (md5) and relevant parameters of each UNode execution. If these factors did not change, re-running your script will not execute the UNode again. This makes it possible to cancel Ursgal scripts and resume them later without losing progress, for instance to add an additional search engine. Information about each run is stored in files ending with .u.json. Since the JSON format is human-readable, these files also act as log files that contain all relevant parameters and file paths.

If you want to force UNodes to re-run each time, you can use the force keyword argument. force = True will ignore all JSON files and re-run the UNode even if the parameters did not change. Another useful keyword argument is 'output_file_name', which allows you to define the name of the UNodes' output file. If you don't specify this argument, Ursgal will automatically generate an appropriate output file name (recommended).

```
search_result = uc.search(
    input_file       = 'my_mass_spec_file.mzML',
    engine           = 'omssa',
    force            = True,
    output_file_name = 'my_omssa_result.csv'
)
```

2.1.7 7. Example Scripts

Now that we covered all the basics of Ursgal, you should be able to write a basic Ursgal script. Make sure to check out the *example scripts* folder ("ursgal/example_scripts") which contains a variety of basic and advanced Ursgal scripts that are ready to execute. Example scripts will automatically download the required files before execution.

These example scripts are a good starting point:

- *simple_example_search.py*
- *target_decoy_generation_example.py*
- *do_it_all_folder_wide.py*

CHAPTER 3

Module structure

3.1 General Structure

Each UNode requires a **resource** and a **wrapper** file to be declared properly. Additionally, the Unode parameters have to be declared in the ursgal**/uparams.py** file, which holds the grouped and universal ursgal parameters including specific translations to the different engine parameters, their description, default values and value types (see schematic overview A below).

3.1.1 Schematic Overview

3.1.2 Resources

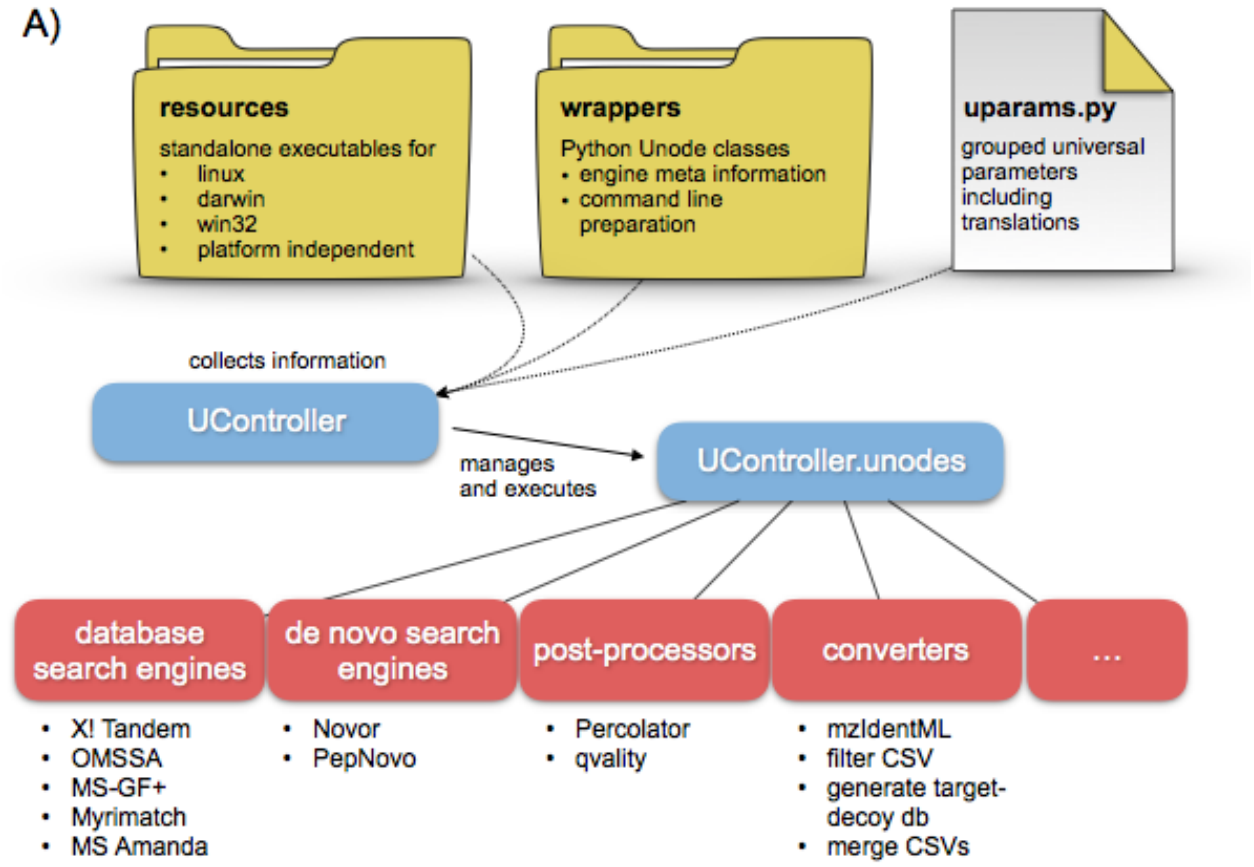
The **resources/** directory contains the main code for each UNode, e.g.:

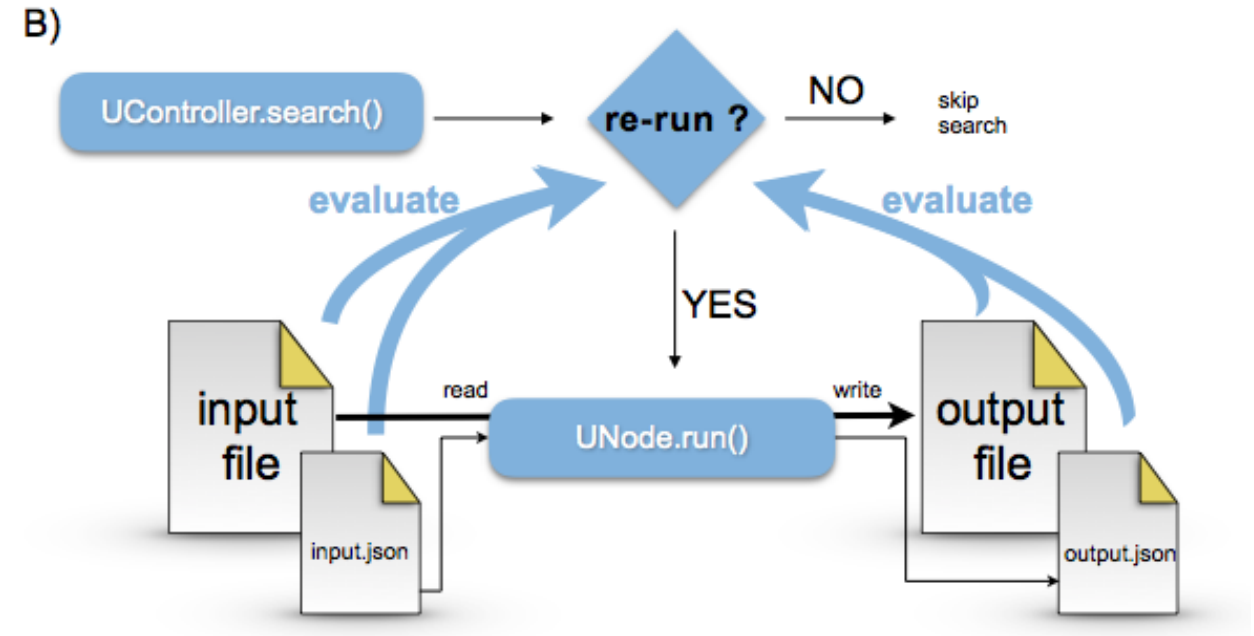
1. executables (i.e. .exe or .jar)
2. standalone Python scripts
3. any additional files that are required by the engine

Compared to the original standalone applications, the folder structure is unchanged. Integration of standalone applications into Ursgal is achieved by Python wrappers around the executables ("wrappers", see below) and entries in the general **ursgal/uparams.py** file.

The resources directory path depends on the platform dependencies of the UNode:

1. <installation path of ursgal>**/resources/**<platform>**/<architecture>** Whereas platform is darwin (OS X), linux or win32 (Windows (and yes even if you have windows 64 bit . . .))
2. Architecture independent engines, like Python scripts or Java packages can be placed in <installation path of ursgal>/resources/platform_independent/arc_independent/
3. Each UNode has to have its own folder following Python class name conventions, **but all lowercase**. For more details in the naming convention see PEP 3131.


A)
resources
standalone executables for
• linux
• darwin
• win32
• platform independent
wrappers
Python Unode classes
• engine meta information
• command line preparation
uparams.py
grouped universal parameters including translations
collects information
UController
manages and executes
UController.unodes
database search engines
de novo search engines
post-processors
converters
...
• X! Tandem
• OMSSA
• MS-GF+
• Myrimatch
• MS Amanda
• Novor
• PepNovo
• Percolator
• qvality
• mzIdentML
• filter CSV
• generate target-decoy db
• merge CSVs


B)
UController.search()
re-run ?
NO
skip search
evaluate
evaluate
YES
input file
read
UNode.run()
write
output file
input.json
output.json

3.1.3 Wrapper Python class

The wrapper inherits from ursgal.UNode. During the instantiation, the default parameters are injected into the class. The default parameters are collected using the umapmaster class, which parses the grouped parameters listed in ursgal.uparams. Therefore, it is imperative that all parameters are listed in the uparams.py file (see below).

The default structure of a wrapper is:

```
#!/usr/bin/env python
import ursgal

class xtandem_alanine( ursgal.UNode ):
    """
    X!Tandem UNode
    Parameter options at http://www.thegpm.org/TANDEM/api/

    Reference:
    Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem mass spectra.
    """
    META_INFO = { ... }

    def __init__(self, *args, **kwargs):
        super(xtandem_alanine, self).__init__(*args, **kwargs)

    def preflight(self):
        # code that should be run before the UNode is executed
        # e.g. writing a config file
        # Note: not mandatory
        return

    def postflight(self):
        # code that should be run after the UNode is executed
        # e.g. formatting the output file
        # Note: not mandatory
        return
```

It is important that the super class is called with the wrapper's name. Default parameters are collected from uparams.py using this name (see below). The special methods *preflight()* and *postflight()* are automatically called by Ursgal's UController when a UNode is launched.

The META INFO

The META_INFO class attributed is most important for proper function. The META_INFO entries are described below; for more examples, please refer to the wrapper folder.

Edit Version

This number is used to determine the most recent version of the wrapper on different work stations. Therefore, it should be updated everytime a change in the wrapper is made.

```
META_INFO = {
    ...
    'edit_version'          : 1.00,
    ...
}
```

Name, Version and Release Date

The original name of the engine, it's version number and the release date of this version (if available).

```
META_INFO = {
    ...
    'name'                : 'X!Tandem',
    'version'             : 'ALANINE',
    'release_date'        : '2017-02-01',
    ...
}
```

Engine Type

Engine Type will define were the engine is grouped into. The groups are shown after ucontroller instantiation. Additionally, the wrapper registers the engines to certain controller functionality, e.g. engine_type['search_engine'] : True will allow ucontroller.search(engine='omssa_2_1_9') to be executed. The engine types and corresponding ucontroller functions are also listed in ukb.ENGINE_TYPES

```
META_INFO = {
    'engine_type' : {
        'controller'                    : False,
        'converter'                     : False,
        'fetcher'                       : False,
        'meta_engine'                   : False,
        'misc_engine'                   : False,
        'cross_link_search_engine'      : False,
        'de_novo_search_engine'         : False,
        'protein_database_search_engine': True,
        'quantification_engine'         : False,
        'validation_engine'             : False,
        'visualizer'                    : False,
    },
    ...
}
```

Citation

Please enter the proper citation for each engine you are wrapping so users can be reminded to cite the proper work. In an academic world, this is the only credit that one can hope for ;) For example.

```
META_INFO = {
    ...
    'citation' : \
        'Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem '\
        'mass spectra.',
    ...
}
```

Input Extensions

List of file extensions that can be used as input files for the engine. For example.

```
META_INFO = {
    ...
    'input_extensions' : ['.mgf', '.gaml', '.dta', '.pkl', '.mzData', '.mzXML'],
    ...
}
```

Output Extensions

List of file extensions generated by the engine. They are required to auto-generate the output file name and to check if an output file was produced. For example:

```
META_INFO = {
    ...
    'output_extensions'      : ['.csv'],
    ...
}
```

Create own folder

This option allows all files and results for this engine to be placed in its own folder. The engine will define the folder name, here omssa_2_1_9. The master switch for all unodes to create their folder (if it is specified in the META_INFO) is the ucontroler param **engines_create_folders**)

```
META_INFO = {
    ...
    'create_own_folder'      : True,
    ...
}
```

In Development

In development flag will hide the wrapper from the controller overview, however the node will be instantiated during start and is therefore nevertheless available.

```
META_INFO = {
    ...
    'in_development'         : False,
    ...
}
```

Include in GIT

The standalone executable can be distributed via the ursgal git.

Note: Big executables are distributed via the ./install_resources.py script, thus refrain overloading ursgal.git too much :)

```
META_INFO = {
    ...
    'include_in_git'        : False,
    ...
}
```

Distributable

This is True if the coresponding standalone executable can be distributed via the ./install_resources.py and False if the standalone executable needs to be downloaded manually.

```
META_INFO = {
    ...
    'include_in_git'        : False,
    ...
}
```

UTranslation Style

Since ursgal translates the general ursgal parameters to engine specific parameters and multiple versions of one engines can be available in ursgal (see e.g. 4+ X! Tandem versions), we define translation styles. Therefore all X! Tandem versions share (up to now) all parameter translation rules, defined as xtandem_style_1. Which translation style is used for which wrapper is defined by this entry in the META info.

```
META_INFO = {
    ...
    'utranslation_style'    : 'omssa_style_1',
    ...
}
```

Download information

The download information is required for the install_resources.py script to function.

```
META_INFO = {
    ...
    ### Below are the download information ###
    'engine': {
        'darwin' : {
            '64bit' : {
                'exe'            : 'omssacl',
                'url'            : 'ftp://ftp.ncbi.nih.gov/pub/lewisg/omssa/2.1.9/
↪omssa-2.1.9.macos.tar.gz',
                'zip_md5'        : '9cb92a98c4d96c34cc925b9336cbaec7',
                'additional_exe' : ['makeblastdb'],
            },
        },
        'linux' : {
            '64bit' : {
                'exe'            : 'omssacl',
                'url'            : 'ftp://ftp.ncbi.nih.gov/pub/lewisg/omssa/2.1.9/
↪omssa-2.1.9.linux.tar.gz',
```

(continues on next page)

(continued from previous page)

```
                'zip_md5'        : '921e01df9cd2a99d21e9a336b5b862c1',
                'additional_exe' : ['makeblastdb'],
            },
        },
        'win32' : {
            '64bit' : {
                'exe'            : 'omssacl.exe',
                'url'            : 'ftp://ftp.ncbi.nih.gov/pub/lewisg/omssa/2.1.9/
↪omssa-2.1.9.win32.exe',
                'zip_md5'        : 'b9d9a8aec3cfe77c48ce0f5752aba8f9',
                'additional_exe' : ['makeblastdb'],
            },
            '32bit' : {
                'exe'            : 'omssacl.exe',
                'url'            : 'ftp://ftp.ncbi.nih.gov/pub/lewisg/omssa/2.1.9/
↪omssa-2.1.9.win32.exe',
                'zip_md5'        : 'a05a5cdd45fd8abcfc75b1236f8a2390',
                'additional_exe' : ['makeblastdb'],
            },
        },
    },
    ...
}
```

3.1.4 Grouped parameters - uparams.py

The ursgal/uparams.py file holds all parameter information available in ursgal. All default parameters for all nodes are stored there, can be accessed and modified. This file contains one Python dictionary with keys representing the ursgal parameter.

Note: The entries 'uvalue_option' and 'edit_version' will be removed with ursgal version 0.7.0. Let us know if and why you require them.

Entries:

- **'available_in_unode'** Defines which nodes use this parameter. Complete engine names are given.
- **'default_value'** Defines the default value for this parameter. Please note that these can be adjusted via parameters or profiles.
- **'description'** Provides a short explanatory text for the parameter.
- **'edit_version'** This number is used to determine the most recent version of the uparam on different work stations. Therefore, it should be updated everytime a change in the wrapper is made.
- **'trigger_rerun'** Defines if a change in this parameter will cause the unode to be executed, independently if there are already result files present. Since not all parameter changes require re-execution, this ensures minimal total runtime for pipelines.
- **'ukey_translation'** Defines how the ursgal parameter name is translated into the name in the corresponding engine. The unified parameter name in ursgal helps the user to group the parameter names from different engines and simplifies the parameter handling for the user.
- **'utag'** Helps to sort and group parameters.

- **'uvalue_translation'** Defines how the ursgal parameter value is translated into the value of the corresponding engines. Please note that the value type can change when its translated, in order to be functional for the engine.
- **'uvalue_type'** Defines the uvalue type of this parameter.
- **'uvalue_option'** Provides informations for parameter settings in the GU, e.g. possible parameter value ranges and step sizes. The required informations depend on the uvalue type and are listed in the following.

uvalue_option entries for different uvalue_types:

- **'bool':** no uvalue_options are required
- **'float'/'int':**

  'none_val': This value will be set to None (since None cannot be entered in the GUI)

  'max': maximal value to be entered

  'min': minimal value to be entered

  'f-point': number ob decimal points (not available/needed for uvalue_type 'int')

  'updownval': step size of user control arrows/slider

  'unit': unit of the uvalue

- **'str':**

  'none_val': This value will be set to None (since None cannot be entered in the GUI)

  'multiple_line': True, if the user should be allowed to enter multiple lines

- **'list':**

  'none_val': This value will be set to None (since None cannot be entered in the GUI)

  'item_title': generic title for the items in the list

  'item_type': type of items in the list,

  'custom_val_max': maximum number of entries (?)

  Please note: depending on the item_type, additional options are required, e.g. multiple_line if the item_type is 'str'

- **'dict':**

  'none_val': This value will be set to None (since None cannot be entered in the GUI)

  'multiple_line': True, if the user should be allowed to enter multiple lines (given for each key in the dict)

  'item_titles': generic titles for the keys and values in the dict (given in the same structure as the dict)

  'value_types': types of values for each key in the dict

  Please note: depending on the value_types, additional options are required, e.g. max/min/f-point/updownval/unit if the item_type is 'float'

- **'select':**

  'select_type': type of selection from available_values, e.g. 'combo_box' if combinations of multiple values can be selected, 'radio_button' if only one value can be selected

  'available_values': list of available values to be selected

  'custom_val_max': maximum number of entries (?)

The following example shows the parameter dict for the 'frag_mass_tolerance' parameter.

```
ursgal_params = {
    'frag_mass_tolerance' : {
        'edit_version' : 1.00,
        'available_in_unode' : [
            'moda_v1_51',
            'msamanda_1_0_0_5242',
            'msamanda_1_0_0_5243',
            'msamanda_1_0_0_6299',
            'msamanda_1_0_0_6300',
            'msamanda_1_0_0_7503',
            'msamanda_1_0_0_7504',
            'msfragger_20170103',
            'myrimatch_2_1_138',
            'myrimatch_2_2_140',
            'novor_1_1beta',
            'omssa_2_1_9',
            'pepnovo_3_1',
            'pipi_1_4_5',
            'pipi_1_4_6',
            'xtandem_cyclone_2010',
            'xtandem_jackhammer',
            'xtandem_piledriver',
            'xtandem_sledgehammer',
            'xtandem_vengeance',
            'xtandem_alanine',
        ],
        'triggers_rerun' : True,
        'ukey_translation' : {
            'moda_style_1'      : 'FragTolerance',
            'msamanda_style_1'  : 'ms2_tol',
            'msfragger_style_1' : 'fragment_mass_tolerance',
            'myrimatch_style_1' : 'FragmentMzTolerance',
            'novor_style_1'     : 'fragmentIonErrorTol',
            'omssa_style_1'     : '-to',
            'pepnovo_style_1'   : '-fragment_tolerance',
            'pipi_style_1'      : 'ms2_tolerance',
            'xtandem_style_1'   : 'spectrum, fragment monoisotopic mass error',
        },
        'utag' : [
            'fragment',
        ],
        'uvalue_translation' : {
        },
        'uvalue_type' : 'int',
        'uvalue_option' : {
            'none_val'  : None,
            'max'       : 100000,
            'min'       : 0,
            'updownval' : 1,
            'unit'      : ''
        },
        'default_value' : 20,
        'description' : \
            'Mass tolerance of measured and calculated fragment ions',
    },
    ...
}
```

CHAPTER 4

Ursgal module contents

4.1 UController

class ursgal.ucontroller.**UController**(**args*, ***kwargs*)

ursgal main class

Keyword Arguments

- **params** (*dict*) – params that are used for all further analyses, overriding default values from ursgal/uparams.py
- **profile** (*str*) – Profiles key for faster parameter selection. This idea is adapted from MS-GF+ and translated to all search engines.

  Currently available profiles are:

  – 'QExactive+'

  – 'LTQ XL high res'

  – 'LTQ XL low res'

Example:

```
>>> us = ursgal.UController(
...     profile = 'LTQ XL low res',
...     params = { 'database': 'BSA.fasta' }
...)
```

combine_search_results(*input_files*, *engine=None*, *force=None*, *output_file_name=None*)

The ucontroller combine_search_results function combines search result .csv files that were generated by different search engines.

Keyword Arguments

- **input_files** (*list*) – A list containing the complete paths to two or more input files. Input files have to be unified result .csv files that were produced by different engines.

- **engine** (*str*) – The name of the desired search result combiner. Can also be a shortened version if it is unambigous.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> uc=ursgal.UController()
>>> unified_merged_results = [
...     'BSA_xtandem_piledriver_unified_merged.csv',
...     'BSA_msgfplus_unified_merged.csv',
...     'BSA_omssa_unified_merged.csv'
...]
>>> uc.combine_search_results(
...     input_files = unified_merged_results,
...     engine      = 'combine_FDR_0_1'
...)
```

Note: If you have multiple result files from the same engine, you can merge them with *merge_csvs()*.

Returns Path of the output file

Return type str

convert (*input_file*, *engine=None*, *force=None*, *output_file_name=None*, *guess_engine=False*)

The UController convert function converts the given input_file into another format as defined by the specified engine.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input file.
- **engine** (*str*) – The name of the desired converter engine. Can also be a shortened version if it is unambigous.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.
- **guess_engine** (*bool*) – The converter engine is guessed based on the input file. This works so far for mzml2mgf conversion and conversion of search_engine result files to csv.

Example:

```
>>> uc=ursgal.UController()
>>> unified_merged_results = 'BSA_msgfplus_unified_merged.csv',
>>> uc.convert_file(
...     input_file = unified_merged_results,
...     engine     = 'csv2ssl_1_0_0'
...)
```

Returns Path of the output file

Return type str

convert_results_to_csv (*input_file*, *force=None*, *output_file_name=None*)
The ucontroller convert_results_to_csv function

Note: uses the Java mzidentml library (Reisinger et al., 2012)

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file currently has to be an identification engine result file
- **force** (*bool*) – (re)do the analysis if output files already exists
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> us=ursgal.UController( profile='LTQ XL high res' )
>>> us.convert_results_to_csv(
...     input_file = 'my_result.xml',
...)
```

Returns Path of the output file

Return type str

Notes: internal function, use *convert()* instead

convert_to_mgf_and_update_rt_lookup (*input_file*, *force=None*, *output_file_name=None*)
Converts the mzML to mgf and updates the scanID to retention time lookup. The looukp is needed for the unifying of the .csv files.

Parameters **input_file** (*str*) – mzML input file name

Returns name of the output mgf file

Return type str

Notes: internal function, use *convert()* instead

determine_availability_of_unodes ()
The ucontroller determine_availability_of_unodes function

Note: internal function

Checks for engines in ursgal/resources/<platform>/<architecture> and expects the executable to be in the corresponding folder.

distinguish_multi_and_single_input (*in_input*)
Finds out whether the input is a single file or a list of files and returns a bool indicating so, as well as the input file(s)

download_resources (*resources=None*)
Function to download all executable from the specified http url

Keyword Arguments **resources** (*list*) – list of specific resources that should be downloaded. If left to None, all possible resources are downloaded.

dump_multi_json (*fpath*, *fdicts*)
For UNodes that take multiple input files. Generates a json for the multi-input helper file. This json allows ursgal to check whether input changed or not, to determine if a node has to be re-run or not.

engine_sanity_check(*short_engine*)
The ucontroller engine_sanity_check function

Takes input and name and tries to guess the full engine name, e.g. including the version number. omssa as inpout will yield omssa_2_1_9 if there is only one omssa engine installed, i.e. the mapping (<stored_fulle_engine_name>.startswith(<input>) has to be unique and defined.

Additionally, sanity check also validates if engine is available on the system.

Note: internal function, since assertion error is called.

Parameters **short_engine** (*str*) – engine short name or tag

calls *self.guess_engine_name()*

Returns Full name of the engine or None.

Return type str

eval_if_run_needs_to_be_executed(*engine=None*, *force=None*)
Returns the reason why self.run needs to be executed or None if there is no need

execute_misc_engine(*input_file*, *engine=None*, *force=None*, *output_file_name=None*, *merge_duplicates=False*)
The UController execute_misc_engine function

This function can be used to execute any misc engine by only giving the input_file and engine name.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, a unified (and possibly merged) search result .csv.
- **engine** (*str*) – the name of the validation engine which should be run, can also be a short version if this name is unambigous
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.
- **merge_duplicates** (*bool*) – If True, the produced output file will be checked for duplicated PSMs, which will be merged into a single line. Caution, the original output file will be overwritten!

Note: Input files to *validate()* must be in unified csv format (i.e. output files of *search()* or *unify_csv()*).

Example:

```
>>> my_databases = ['homo_sapiensA.fasta', 'homo_sapiensB.fasta']
>>> uc = ursgal.UController()
>>> new_target_decoy_db = uc.execute_misc_engine(
...     input_files      = my_databases,
...     engine           = 'generate_target_decoy_1_0_0',
...     output_file_name = 'my_homo_sapiens_target_decoy_db.fasta'
...)
```

Returns Path of the output file

Return type str

execute_unode(*input_file*, *engine=None*, *force=False*, *output_file_name=None*, *dry_run=False*, *merge_duplicates=False*)

The UController execute_unode function. Executes arbitrary UNodes, as specified by their name.

Keyword Arguments

- **input_file** (*str or list of str*) – The complete path to the input, or a list of paths to the input files.
- **engine** (*str*) – Engine name one wants to execute
- **force** (*bool*) – (Re)do the analysis if output files already exists
- **dry_run** (*bool*) – Do not execute; only return the output file name

---

Note: Can also execute UNodes that are tagged as 'in development' in kb (=not shown in UController overview) if their name is specified.

---

fetch_file(*engine=None*)

The UController fetch_file function

Downloads files (FTP or HTTP).

Keyword Arguments **engine** (*str*) – Available options are 'get_http_files_1_0_0' and 'get_ftp_files_1_0_0'

Example:

```
>>> params = {
...     'ftp_url'       : 'ftp.peptideatlas.org',
...     'ftp_login'         : 'PASS00269',
...     'ftp_password'      : 'FI4645a',
...     'ftp_include_ext'   : [
...         'JB_FASP_pH8_2-3_28122012.mzML',
...     ],
...     'ftp_output_folder' : '/home/Desktop/,
... }
>>> uc = ursgal.UController(
...     params = params
... )
>>> uc.fetch_file(
...     engine     = 'get_ftp_files_1_0_0'
... )
```

Returns Path of the downloaded file

Return type str

filter_csv(*input_file*, *force=False*, *output_file_name=None*)

[WARNING] This function is not supported anymore! Please use *execute_misc_engine()* instead

The UController filter_csv function

Filters .csv files row-wise according to user-defined rules.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file has currently to be a .csv file.

- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

The filter rules have to be defined in the params. See the engine documentation for further information (*filter_csv_1_0_0._execute()*).

Example

```
>>> # Only columns with these attributes will be retained:
>>> # a) 'PEP' column value must be lower than or equal to 0.01
>>> # b) 'Is decoy' column value must equal 'false'
>>> uc.params['csv_filter_rules'] = [
...     ['PEP',      'lte',    0.01   ],
...     ['Is decoy', 'equals', 'false']
... ]
>>> uc.filter_csv( 'my_results.csv' )
```

generate_multi_file_dicts (*input_files*)

generates a file_dict for access in the UNode classes. in the UNode classes, a file_dict can be found for each input file under self.params["input_file_dicts"]. also adds some "quick-access" entries to the file_dicts. these file dicts contain the input/output file dicts for that file, as well as quick-access information (i.e. "last_engine")

generate_multi_helper_file (*input_files*)

for UNodes that take multiple input files. generates a temporary single input helper file, which acts as the input file so that all the routines (set_io, write history) work normally with multiple files.

generate_target_decoy (*input_files=None*, *engine=None*, *force=False*, *output_file_name=None*)

[WARNING] This function is not supported anymore! Please use *execute_misc_engine()* instead

The ucontroller function for target_decoy database generation.

Keyword Arguments

- **input_files** (*list*) – List with complete paths to one or more fasta databases.
- **engine** (*str*) – name of the database generator which should be run, can also be a short version if this name is unambigous
- **force** (*bool*) – (re)do the analysis if ouput files already exists
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> my_databases = ['homo_sapiensA.fasta', 'homo_sapiensB.fasta']
>>> uc = ursgal.UController()
>>> new_target_decoy_db = uc.generate_target_decoy(
...     input_files      = my_databases,
...     engine           = 'generate_target_decoy_1_0_0',
...     output_file_name = 'my_homo_sapiens_target_decoy_db.fasta'
...)
```

The returned database can then be set as the new database for searches.

Example:

```
>>> uc.params['database'] = new_target_decoy_db
```

Returns Name/path of the output file

Return type str

get_mzml_that_corresponds_to_mgf(*mgf_path*)
Checks the history of a MGF file to determine which mzML is stems from. Returns the path to that mzML.

guess_engine_name(*short_engine*)
The ucontroller function for guessing the right engine name from a short name. For example 'omssa' is translated into omssa_2_1_9 which is the only available version of omssa in ursgal. If you use an ambigous name or if a engine has multiple version, it is required to name the engine unambigously. Instead of myrimatch use myrimatch_2_1_138.

Parameters **short_engine** (*str*) – engine short name or tag

Iterates over *self.unodes.keys()* and checks if:

- the keys start with the short_engine
- that the match is unique

Notes: internal function

Returns

Full name of engine or *None* if short_engine has multiple hits

Return type str

input_file_sanity_check(*input_file*, *engine=None*, *extensions=None*, *multi=False*, *custom_str=None*)
The ucontroller input_file_sanity_check function

Asserts that input files exist, can be read, have the right file type and file extension etc. Raises an AssertionError if any criterion is violated.

Keyword Arguments

- **input_file** (*str or list*) – input file path to be checked, or a list of input file paths in the case of multi-nodes
- **engine** (*str*) – the name of the engine, file extension requirements will be looked up in engine/kb (optional)
- **extensions** (*list*) – a list of permitted file extensions (optional)
- **multi** (*bool*) – whether the UNode accepts multiple input files or not

---

Note: Internal Function

Returns None

map_peptides_to_fasta(*input_file*, *force=False*, *output_file_name=None*)

[WARNING] This function is not supported anymore! Please use *execute_misc_engine()* instead

The ucontroller function to call the upeptide_mapper node.

Note: Different converter versions can be used (see parameter 'peptide_mapper_converter_version') as well as different classes inside the converter node (see parameter 'peptide_mapper_class_version')

Available converter nodes

- upeptide_mapper_1_0_0

Available converter classes of upeptide_mapper_1_0_0

- UPeptideMapper_v3 (default)
- UPeptideMapper_v4 (no buffering and enhanced speed to v3)
- UPeptideMapper_v2

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file has currently to be a .csv file.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Returns Path of the output file

Return type str

merge_csvs(*input_files*, *force=None*, *output_file_name=None*, *merge_duplicates=False*)

The ucontroller merge_csvs function

Merges unified .csv files generated by the same search engine into a single .csv file. This is needed if you want to validate search results from the same identification engine on multiple mzML files. For example if multiple fraction of the original sample for LS-MS/MS analysis were measured and represent a sample/analysis entity.

Keyword Arguments

- **input_files** (*list*) – A list containing the complete paths to two or more input files. Input files have to be .csv files.
- **force** (*bool*) – (re)do the analysis if output file already exists
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> us = ursgal.UController()
>>> xtandem_results = [
...     'BSA_1_xtandem_sledgehammer_unified.csv',
...     'BSA_2_xtandem_sledgehammer_unified.csv',
...     'BSA_3_xtandem_sledgehammer_unified.csv'
... ]
>>> us.merge_csvs( input_files = xtandem_results )
```

Returns Path of the output file

Return type str

quantify(*input_file*, *engine*, *force=None*, *output_file_name=None*, *multi=False*)
The ucontroller quantify function

Performs a peptide/protein quantification using the specified quantification engine and mzML/ident file file. Produces a CSV file with peptide/protein quants in the unified Ursgal CSV format. see: *List of available engines*

Keyword Arguments

- **input_file** (*str*) – The complete path to the mzML file.
- **engine** (*str*) – The name of the quantification engine which should be used, can also be a short version if this name is unambigous.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> uc = ursgal.UController(
...     profile = 'LTQ XL high res',
...     params  = {'evidence': 'BSA_idents.csv'}
... )
>>> uc.quantify(
...     input_file = 'BSA.mzML',
...     engine     = 'pyQms_0_0_1'
... )
```

Returns Path of the output file (unified CSV format)

Return type str

run_unode_if_required(*force*, *engine_name*, *answer*, *merge_duplicates=False*, *history_addon=None*)
The ucontroller run_unode_if_required function

Note: internal function

Executes a UNode if required. Otherwise prints why the run was not required. If the UNode is executed, the corresponding json is dumped and the history is updated.

Keyword Arguments

- **force** (*bool*) – (re)do the analysis if output files already exists
- **engine_name** (*str*) – name of the engine to be executed (after verifying with engine_sanity_check)
- **answer** (*str or None*) – The answer of prepare_unode_run(). Can be None if no re-run is required, or a string indicating the reason for re-run

sanitize_userdefined_output_filename(*user_fname*, *engine*)
If the user defined a node output file name, we remove all path info from it (not supported) and throw a warning; possibly add a prefix; possibly add the correct file extension (if user didn't already include it)

search (*input_file*, *engine=None*, *force=None*, *output_file_name=None*, *multi=False*)
The ucontroller search function

Performs a peptide search using the specified search engine and mzML file. Produces a CSV file with peptide spectrum matches in the unified Ursgal CSV format. see: *List of available engines*

Keyword Arguments

- **input_file** (*str*) – The complete path to the mzML file, or an MGF file that was converted from mzML.
- **engine** (*str*) – The name of the identification engine which should be used, can also be a short version if this name is unambigous.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example::

```
>>> uc = ursgal.UController(
...     profile = 'LTQ XL high res',
...     params  = {'database': 'BSA.fasta'}
... )
>>> uc.search(
...     input_file = 'BSA.mzML',
...     engine     = 'omssa'
... )
```

Returns Path of the output file (unified CSV format)

Return type str

Note: Some search engines require a lot of RAM (up to 14GB, depending on your input files). If you don't have a lot of RAM, some engines might crash. Consider using X!Tandem or OMSSA in these cases, since they are less demanding.

Note: This function calls five search-related ursgal functions in succession, all of which can also be called individually:

- *convert()* (mzml to mgf, if required, using the mzml2mgf engine)
- *search_mgf()*
- *convert()* (raw search results to csv, if required)
- *execute_misc_engine()* (peptide_mapper)
- *execute_misc_engine()* (unify_csv)

---

search_mgf (*input_file*, *engine=None*, *force=None*, *output_file_name=None*, *multi=False*)
The UController search_mgf function

Does the main peptide identification search with the specified identification engine. This function is called with every mzML and every search which should be used. The function uses *UNode.run()* to execute a single search engine. For example to execute X!Tandem via command line.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file has to be a .MGF file (but .mzML files can be converted to .MGF with Ursgal)
- **engine** (*str*) – the name of the identification engine which should be run, can also be a short version if this name is unambigous.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> uc = ursgal.UController(
...     profile ='LTQ XL high res',
...     params  = {'database': 'BSA.fasta'}
... )
>>> uc.search_mgf(
...     input_file = 'BSA.mgf',
...     engine     = 'xtandem_piledriver'
... )
```

Returns Path of the output file

Return type str

Note: Consider using *search()* instead. *search()* automatically converts mzML to MGF and produces a unified CSV output file.

---

set_file_info_dict(*in_file*)
Splits ext and path and so on

set_profile(*profile*, *dev_mode=False*)
The ucontroller set_profile function

Note: internal function

Parameters **profile** (*str*) – Profile speficied to use for all searches.

Available profiles:

- 'QExactive+'
- 'LTQ XL high res'
- 'LTQ XL low res'

Sets self.params according to profile name defined in ursgal.kb.profiles

Example:

```
>>>'LTQ XL low res' : {
...     # MS 1 orbitrap & MSn iontrap
...     'frag_mass_tolerance'       : 0.5,
...     'frag_mass_tolerance_unit'  : 'da',
```

(continues on next page)

(continued from previous page)

```
...     'instrument'                : 'low_res_LTQ',
...     'frag_method'               : 'cid'
...}
```

Own profiles can easily be defined in profiles.py in ursgal/kb according to the need parameters or machine specifications.

show_unode_overview()
The ucontroller show_unode_overview function

Note: internal function

Prints the overview of all available nodes. The overview includes the category, name and availability of each node. Available nodes are highlighted. Here also the correct functionality of the engine avaibility and installation is verified.

unify_csv(*input_file*, *force=False*, *output_file_name=None*)

[WARNING] This function is not supported anymore! Please use *execute_misc_engine()* instead

The ucontroller unify_csv function

Unifies the .csv files which were converted by the mzidentml library. The corrections for each engine are listed in the node under ursgal/resources/arc_independent/unify_csv_1_0_0

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file has currently to be a .csv file.
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> uc=ursgal.UController(
...     profile = 'LTQ XL low res',
...     params  = {'database': 'BSA.fasta'}
... )
>>> xtandem_result_xml = uc.search_mgf(
...     input_file = 'BSA.mzML',
...     engine     = 'xtandem',
... )
>>> xtandem_result_csv = uc.convert_results_to_csv(
...     input_file = xtandem_result_xml
... )
>>> unified_csv = uc.unify_csv(
...     input_file = xtandem_result_csv
... )
```

Returns Path of the output file

Return type str

validate(*input_file*, *engine=None*, *force=None*, *output_file_name=None*)
The UController validate function

Does statistical post-processing of unified search result .csv files with the specified validation engine.

Depending on the validation method a posterior error probability (PEP) and/or a q-value will be available in the final results.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, a unified (and possibly merged) search result .csv.
- **engine** (*str*) – the name of the validation engine which should be run, can also be a short version if this name is unambigous
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.
- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Note: Input files to *validate()* must be in unified csv format (i.e. output files of *search()* or *unify_csv()*).

Example:

```
>>> uc = ursgal.UController(
...     profile = 'LTQ XL low res',
...     params  = {'database': 'BSA.fasta'}
... )
>>> xtandem_result_csv = uc.search(
...     input_file = 'BSA.mzML',
...     engine     = 'xtandem_piledriver'
... )
>>> validated_csv = uc.validate(
...     input_file = xtandem_result_csv,
...     engine     = 'percolator_2_08'
... )
```

Returns Path of the output file

Return type str

verify_engine_produced_an_output_file(*expected_fpath*, *engine_name*)
Since not all engines raise an exception when they fail, we check if the output file was successfully produced or not to throw a proper exception in case the engine crashed.

visualize(*input_files*, *engine=None*, *force=None*, *output_file_name=None*, *multi=True*)
The ucontroller function for visualization

Does graphical visualization of result .csv files.

Keyword Arguments

- **input_files** (*list*) – list with complete paths of .csv files
- **engine** (*str*) – the name of the visualizer which should be run, can also be a short version if this name is unambigous
- **force** (*bool*) – (Re)do the analysis, even if output file already exists.

- **output_file_name** (*str or None*) – Desired output file name excluding path (optional). If None, output file name will be auto-generated.

Example:

```
>>> uc = ursgal.UController( profile='LTQ XL high res' )
>>> xtandem_result_csv = uc.search(
...     input_file = 'BSA.mzML',
...     engine     = 'xtandem_piledriver',
... )
>>> omssa_result_csv = uc.search(
...     input_file = 'BSA.mzML',
...     engine     = 'omssa',
... )
>>> uc.visualize(
...     input_files = [xtandem_result_csv, omssa_result_csv],
...     engine      = 'venndiagram',
... )
```

Note: For detailed information about the VennDiagram UNode, see *venndiagram_1_0_0._execute()*.

Returns Path of the output file

Return type str

4.2 UNode

class ursgal.**UNode**(**args*, ***kwargs*)

ursgal class

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

__weakref__

list of weak references to the object (if defined)

_execute()

The _execute unode function

Executes the unode executable via shell.

Note: internal function Unodes that do not require execution via shell redefine the _execute() function in their engine class.

Returns None

_group_psms(*input_file*, *validation_score_field=None*, *bigger_scores_better=None*)

Reads an input csv and returns a defaultdict with the spectrum title mapping to a sorted list of tuples containing each a) score (from validation_score_field) and b) the whole line dict

Keyword Arguments

- **validation_score_field** (*str*) – fieldname of the column that should be used as validation score for sorting of PSMs. If None, get_last_search_engine is used to get the validation_score_field defined for the last used search engine.

- **bigger_scores_better** (*bool*) – defines if in the validation score are increasing (True) or decreasing (False) with their quality. If None, get_last_search_engine is used to get bigger_scores_better defined for the last used search engine.

abs_paths_for_specific_keys(*params*, *param_keys=None*)
Absolute paths for specific keys from the params dict are determined

Returns params with paths in abspath version

Return type dict

calc_md5(*input_file*)
Calculated MD5 for input_file

Parameters **input_file** (*str*) – Path to file

Returns MD5 of input file

Return type str

Thanks Raymond :) http://stackoverflow.com/questions/7829499/using-hashlib-to-compute-md5-digest-of-a-file-in-python-

collect_and_translate_params(*params*)
Translates ursgal parameters into uNode specific syntax.

1) **Each unode.USED_SEARCH_PARAMS contains params that have** to be passed to the uNode.
2) params values are not translated is they [] or {}
3) params values are translated using:

```
uNode.USEARCH_PARAM_VALUE_TRANSLATIONS
> translating only values, regardless of key
uNode.USEARCH_PARAM_KEY_VALUE_TRANSLATOR
> translating only key:value pairs to key:newValue
```

Those lookups are found in kb/{engine}.py

TAG:

- v0.4

compare_json_and_local_ursgal_version(*history*, *json_path*)
Print a warning if the history is a from a different version number

determine_common_name(*input_files*, *mode=None*)
The unode function determines for a list of input files a basic common name

Keyword Arguments **mode** – head or tail for first or last part of the filename, respectively

Parameters **input_files** (*list*) – list with input file names

Returns common file name

Return type str

determine_common_top_level_folder(*input_files=None*)
The unode function determines for a list of input files a common top level folder they all belong to

Keyword Arguments **input_files** (*list*) – list with input files

Returns The common top level folder

Return type str

dump_json_and_calc_md5 (*stats=None*, *params=None*, *calc_md5=True*)
Dumps json with params and stats and calcs md5 for output

Deletes all entries that are defined in params['del_from_params_before_json_dump'] or keys that start with '_'

fix_md5_and_file_in_json (*ifile=None*, *json_path=None*)
Fixes supplementary output json

flatten_list (*multi_list=[]*)
The unode get_last_engine function

Reduces a multidimensional list of lists to a flat list including all elements

get_last_engine (*history=None*, *engine_types=None*, *multiple_engines=False*)
The unode get_last_engine function

Note: returns None if the specified engine type was not used yet.

Keyword Arguments

- **history** (*list*) – A list of path unodes, timestamps and parameters that were used. This function can be used on the history loaded from a file .json, to find out which search engine was used on that file. If not specified, this information is taken from the unode class itself, and not a specific file.
- **engine_types** (*list*) – the engine type(s) for which the last used engine should be identified
- **multiple_engines** (*bool*) – if muliple engines have been used, this can be set to True. Then reports a list of used engines.

Examples

```
>>> fpaths = self.generate_basic_file_info( "14N_xtandem.csv" )
>>> file_info, __     = self.load_json( fpaths=fpaths, mode='input')
>>> last_engine       = self.get_last_engine(
        history      = file_info["history"],
        engine_types = ["protein_database_search_engine"]
    )
>>> print( last_engine )
"xtandem_sledgehammer"
```

Returns The name of the last engine that was used.

Return type str

get_last_engine_type (*history=None*)
The unode get_last_engine_type function

Keyword Arguments **history** (*list*) – A list of path unodes, timestamps and parameters that were used. This function can be used on the history loaded from a file .json, to find out which search engine was used on that file. If not specified, this information is taken from the unode class itself, and not a specific file.

Examples

```
>>> fpaths = self.generate_basic_file_info( "14N_xtandem.csv" )
>>> file_info, __     = self.load_json( fpaths=fpaths, mode='input')
>>> last_engine_type = self.get_last_engine_type(
        history      = file_info["history"],
    )
>>> print( last_engine_type )
"protein_database_search_engine"
```

Returns The type of the last engine that was used. Returns None if the engine_type cannot be specified or if no engine was previously executed on this file.

Return type str

get_last_search_engine(*history=None*, *multiple_engines=False*)
The unode get_last_search_engine function

Note: returns None if no search engine was not used yet.

Keyword Arguments

- **history** (*list*) – A list of path unodes, timestamps and parameters that were used. This function can be used on the history loaded from a file .json, to find out which search engine was used on that file. If not specified, this information is taken from the unode class itself, and not a specific file.
- **multiple_engines** (*bool*) – if muliple engines have been used, this can be set to True. Then reports a list of used engines.

Examples

```
>>> fpaths = self.generate_basic_file_info( "14N_xtandem.csv" )
>>> file_info, __ = self.load_json( fpaths=fpaths, mode='input')
>>> last_engine   = self.get_last_search_engine(
        history   = file_info["history"]
    )
>>> print( last_engine )
"xtandem_sledgehammer"
```

Returns The name of the last search engine that was used. Returns None if no search engine was used yet.

Return type str

import_engine_as_python_function(*function_name=None*)
The unode import_engine_as_python_function function

Imports the main function from a unodes "executable". For unodes that are written completely in python and can be executed by importing them instead of using the command line.

Examples

```
>>> us = ursgal.UController()
>>> cFDR_unode = us.unodes["combine_FDR_0_1"]["class"]
>>> cFDR_main  = cFDR_unode.import_engine_as_python_function()
>>> cFDR_main(
    input_file_list = ["1.csv", "2.csv"],
    directory       = "/tmp/",
)
```

Returns The function called “main” that is specified in the engines python script.

Return type function

Note: Assertion exception if the executable is not a python script, or has no main function.

map_mods()

Maps modifications defined in params[“modification”] using unimod.

Examples

```
>>> [
...     "M,opt,any,Oxidation",          # Met oxidation
...     "C,fix,any,Carbamidomethyl",    # Carbamidomethylation
...     "*,opt,Prot-N-term,Acetyl"      # N-Acteylation
... ]
```

peptide_regex(*database*, *protein_id*, *peptide*)

Note: This function is not longer used at the moment.

The unode peptide_regex function

Parameters

- **database** (*str*) – Name of the used fasta database
- **protein_id** (*str*) – protein ID of the processed protein
- **peptide** (*str*) – peptide which should be mapped on the protein ID’s sequence

This function takes a peptide sequence and maps it to its according proteins sequence, returning the start and stop position in the sequence as well as the amino acid before and after the peptide sequence in the full protein sequence. If the peptide sequence contains known amino acid substitutions like U (Selenocystein) or J (Leucin or Isoleucin) this amino acid is replaced by a regex wildcard ‘.’ in order to be matchable on the fasta database (this is defined in kb.unify_csv_1_0_0.py). This is especially needed if the original sequence contains a ‘X’ and the search engine guesses/determines the amino acid at this position.

If the protein ID is ambigous, the peptide is matched against all protein candidates and the positions, pre- and post aminoacids in the matching sequence as well as the full protein ID as named in the fasta database is returned. This is especially needed for MS Amanda results where protein IDs are returned truncated and become ambigous for some databases.

If the peptide occurs several times in the protein, all occurences are returned.

The function uses a buffer to perform the regex only once for (peptide, protein, database) tuples. All fasta sequences are also buffered in *self.lookups['fasta_dbs']* with the name of the database as key and then all protein IDs and sequnces as key, value pairs.

Pre and post amino acids are required for e.g. percolator input files.

Note: The regex and peptide to protein ID mapping may take a while, if a large file has to be processed.

Returns list of tuples [(peptide_start, peptide_stop, aa_before_peptide, aa_after_peptide, protein_id)]

Return type list

postflight()
This can be/is overwritten by the engine uNode class

preflight()
This can be/is overwritten by the engine uNode class

run(*json_path=None*)
The general run function.

Runs engine/uNode child with given params on defined input_file. This function is automatically called by all ucontroller functions that take an input file and produce a single output file (i.e. ucontroller.search() and ucontroller.validate())

Keyword Arguments

- **json_path** (*str*) – path to input file json, dumped by a controller
- **input_file** (*#*) – path to the input file
- **fpaths** (*#*) – dictionary containing file path information.
- **If None, this is generated using unode.generate_basic_file_info** (*#*) –
- **force** (*#*) – (re)do the analysis if ouput files already exists

Returns Report of the run.

Return type dict

Note: Internal function. This function executes the preflight, postflight and _execute functions, if defined in the engine python script.

---

time_point(*tag=None*, *diff=True*, *format_time=False*, *stop=False*)
Stores time_points in self.stats['time_points'] given a tag. returns time since tag was inserted if tag already exists.

update_output_json()
Updates self.io['output']['params'] with self.io['input']['params']

Although re-run might not be triggered, we need to update the output_json.

update_params_with_io_data()
Generates a flat structure in params combining io.['input']['finfo'] & io.['output']['finfo']

4.3 UCore

ursgal.ucore.**calculate_mass**(*mz*, *charge*)
Calculate mass function

Keyword Arguments

- **mz** (*float*) – mz of molecule/peak
- **charge** (*int*) – charge for calculating mass

Returns calculated mass

Return type float

ursgal.ucore.**calculate_mz**(*mass*, *charge*)
Calculate m/z function

Keyword Arguments

- **mass** (*float*) – mass for calculating m/z
- **charge** (*int*) – charge for calculating m/z

Returns calculated m/z

Return type float

ursgal.ucore.**convert_dalton_to_ppm**(*da_value*, *base_mz=1000.0*)
Convert the precision in Dalton to ppm

Keyword Arguments

- **da_value** (*float*) – Dalton value to transform
- **base_mz** (*float*) – factor for transformation

Returns value in ppm

Return type float

ursgal.ucore.**convert_ppm_to_dalton**(*ppm_value*, *base_mz=1000.0*)
Normalize the precision in ppm to 1000 Dalton

Keyword Arguments

- **ppm_value** (*float*) – parts per million value to transform
- **base_mz** (*float*) – factor for transformation

Returns value in Dalton

Return type float

ursgal.ucore.**count_distinct_psms**(*csv_file_path=None*, *psm_defining_colnames=None*)
Returns a counter based on PSM-defining column names (i.e spectrum & peptide, but also score field because sometimes the same PSMs are reported with different scores. . .).

ursgal.ucore.**digest**(*sequence*, *enzyme*, *count_missed_cleavages=None*, *no_missed_cleavages=False*)
Amino acid digest function

Keyword Arguments

- **sequence** (*str*) – amino acid sequence to digest

- **enzyme** (*tuple*) – enzyme properties used for cleavage ('aminoacid(s)', 'N/C(terminus)') e.g. ('KR','C') for trypsin
- **count_missed_cleavages** (*int*) – number of miss cleavages allowed

Returns list of digested peptides

Return type list

ursgal.ucore.**merge_duplicate_psm_rows**(*csv_file_path=None*, *psm_counter=None*, *psm_defining_colnames=None*, *psm_colnames_to_merge_multiple_values={}*, *joinchar='<|>'*, *overwrite_file=True*)

Rows describing the same PSM (e.g. when two proteins share the same peptide) are merged to one row.

ursgal.ucore.**merge_rowdicts**(*list_of_rowdicts*, *psm_colnames_to_merge_multiple_values*, *joinchar='<|>'*)

Merges CSV rows. If the column values are conflicting, they are joined with a character (joinchar).

ursgal.ucore.**parse_fasta**(*io*)

Small function to efficiently parse a file in fasta format.

Keyword Arguments **io** (*obj*) – openend file obj (fasta file)

Yields *tuple* – fasta_id and sequence

ursgal.ucore.**print_current_params**(*params*, *old_params=None*)

Function to print current params

Keyword Arguments **params** (*dict*) – parameter dict to print

ursgal.ucore.**reformat_peptide**(*regex_pattern*, *unimod_name*, *peptide*)

reformats the MQ and Novor peptide string to ursgal format (ac)SSSLM(ox)RPGPSR –> SSSLMRPGPSR#Acetyl:0;Oxidation:5

ursgal.ucore.**terminal_supports_color**()

Returns True if the running system's terminal supports color, and False otherwise. Source: https://github.com/django/django/blob/master/django/core/management/color.py

4.4 UMapMaster

Mapping classes of Ursgal

4.4.1 UParamMapper

class ursgal.umapmaster.**UParamMapper**(**args*, ***kwargs*)

UParamMapper class offers interface to ursgal.uparams

By default, the ursgal.uparams are parsed and the UParamMapper class is set with the ursgal_params dictionary.

get_masked_params(*mask=None*)

Lists all uparams and the fields specified in the mask

For example:

```
upapa.get_masked_params( mask = ['uvalue_type']) will return::
    {
        '-xmx' : {
```

(continues on next page)

(continued from previous page)

```
            'uvalue_type' : "str",
        },
        'aa_exception_dict' : {
            'uvalue_type' : "dict",
        },
        ...
    }
```

group_styles(*engine=None*)

Parses self.items() and build up lookups. Consistency check (to guarantee that each engine is mapping only on one style) is no longer performed here, but in ucontroller._collect_all_unode_wrappers() instead. engine_2_style and style_2_engine translations are also not built here anymore, but collected from the wrappers (just setting placeholders here).

The lookup build and returned looks like:

```
lookup = {
    'style_2_engine' : {
        'xtandem_style_1' : [
            'xtandem_sledgehamer',
            'xtandem_cylone',
            ...
        ],
        'omssa_style_1' ...
    },
    # This is done during uNode initializations
    # each unode will register its style with umapmaster
    #
    'engine_2_style' : {
        'xtandem_sledgehamer' : 'xtandem_style_1', ...
    },
    'engine_2_params' : {
        'xtandem_sledgehamer' : [ uparam1, uparam2, ...], ...
    },
    'style_2_params' : {
        'xtandem_style_1' : [ uparam1, uparam2, ... ], ...
    },
    'params_triggering_rerun' : {
        'xtandem_style_1' : [ uparam1, uparam2 .... ]
    }
}
```

mapping_dicts(*engine_or_engine_style*, *ext_lookup={}*)

yields all mapping dicts

4.5 Chemical Composition

class ursgal.**ChemicalComposition**(*sequence=None*, *aa_compositions=None*, *isotopic_distributions=None*, *monosaccharide_compositions=None*)

Chemical composition class. The actual sequence or formula can be reset using the *add* function.

Keyword Arguments

- **sequence** (*str*) – Peptide or chemical formula sequence

- **aa_compositions** (*Optional[dict]*) – amino acid compositions
- **isotopic_distributions** (*Optional[dict]*) – isotopic distributions

Keyword argument examples:

sequence - Currently this can for example be:: ['+H2O2H2-OH', '+{0}'.format('H2O'), '{peptide}'.format(pepitde='ELVISLIVES'), '{peptide}+{0}'.format('PO3', peptide='ELVISLIVES'), '{peptide}#{unimod}:{pos}'.format(peptide = 'ELVISLIVES', unimod = 'Oxidation', pos = 1)]

Examples::

```
>>> c = ursgal.ChemicalComposition()
>>> c.use("ELVISLIVES#Acetyl:1")
>>> c.hill_notation()
'C52H90N10O18'
>>> c.hill_notation_unimod()
'C(52)H(90)N(10)O(18)'
>>> c
{'O': 18, 'H': 90, 'C': 52, 'N': 10}
>>> c.composition_of_mod_at_pos[1]
defaultdict(<class 'int'>, {'O': 1, 'H': 2, 'C': 2})
>>> c.composition_of_aa_at_pos[1]
{'O': 3, 'H': 7, 'C': 5, 'N': 1}
>>> c.composition_at_pos[1]
defaultdict(<class 'int'>, {'O': 4, 'H': 9, 'C': 7, 'N': 1})
```

```
>>> c = ursgal.ChemicalComposition('+H2O2H2')
>>> c
{'O': 2, 'H': 4}
>>> c.subtract_chemical_formula('H3')
>>> c
{'O': 2, 'H': 1}
```

Note: We did not include mass calculation, since pyQms will calculate masses much more accurately using unimod and other element enrichments.

---

add_chemical_formula(*chemical_formula*, *factor=1*)

Adds chemical formula to the instance

Parameters **chemical_formula** (*str*) – chemical composition given as Hill notation

Keyword Arguments **factor** (*int*) – multiplication factor to add the same chemical formula multiple times

add_glycan(*glycan*)

Adds a glycan to the instance.

Parameters **glycan** (*str*) – sequence of monosaccharides given in unimod format, e.g.: HexNAc(2)Hex(3)dHex(1)Pent(1), available monosaccharides are listed in chemical_composition_kb

add_peptide(*peptide*)

Adds peptide sequence to the instance

clear()

Resets all lookup dictionaries and self

One class instance can be used analysing a series of sequences, thereby avoiding class instantiation overhead

composition_at_pos = None
chemical composition at given peptide position incl modifications (if peptide sequence was used as input or using the *use* function)

Note: Numbering starts at position 1, since all PSM search engines use this nomenclature.

Type dict

composition_of_aa_at_pos = None
chemical composition of amino acid at given peptide position (if peptide sequence was used as input or using the *use* function)

Note: Numbering starts at position 1, since all PSM search engines use this nomenclature.

Type dict

composition_of_mod_at_pos = None
chemical composition of unimod modifications at given position (if peptide sequence was used as input or using the *use* function)

Note: Numbering starts at position 1, since all PSM search engines use this nomenclature.

Type dict

hill_notation(*include_ones=False*, *cc=None*)
Formats chemical composition into Hill notation string.

Parameters **cc** (*dict, optional*) – can format other element dicts as well.

Returns

Hill notation format of self.

For examples:: C50H88N10O17

Return type str

hill_notation_unimod(*cc=None*)
Formats chemical composition into Hill notation string adding unimod features.

Parameters **cc** (*dict, optional*) – can format other element dicts as well.

Returns

Hill notation format including unimod format rules of self.

For example:: C(50)H(88)N(10)O(17)

Return type str

subtract_chemical_formula(*chemical_formula*, *factor=1*)
Subtract chemical formula from instance

Parameters **chemical_formula** (*str*) – chemical composition given as Hill notation

Keyword Arguments **factor** (*int*) – multiplication factor to add the same chemical formula multiple times

subtract_peptide (*peptide*)
Subtract peptide from instance

use (*sequence*)
Re-initialize the class with a new sequence

This is helpful if one wants to use the same class instance for multiple sequence since it remove class instantiation overhead.

Parameters **sequence** (*str*) – See top for possible input formats.

4.6 Unimod Mapper

class ursgal.**UnimodMapper**
UnimodMapper class that creates lookup to the unimod.xml and userdefined_unimod.xml found located in ursgal/kb/ext and offers several helper methods described below :

appMass2element_list (*mass*, *decimal_places=2*)
Creates a list of element composition dicts for a given approximate mass

Parameters **mass** (*float*) –

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Dicts of elements

Return type list

Examples::

```
>>> import ursgal
>>> U = ursgal.UnimodMapper()
>>> U.appMass2element_list(18, decimal_places=0)
[{'F': 1, 'H': -1}, {'13C': 1, 'H': -1, '2H': 3},
    {'H': -2, 'C': -1, 'S': 1}, {'H': 2, 'C': 4, 'O': -2},
    {'H': -2, 'C': -1, 'O': 2}]
```

appMass2id_list (*mass*, *decimal_places=2*)
Creates a list of unimod ids for a given approximate mass

Parameters **mass** (*float*) –

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Unimod IDs

Return type list

Examples::

```
>>> import ursgal
>>> U = ursgal.UnimodMapper()
>>> U.appMass2id_list(18, decimal_places=0)
['127', '329', '608', '1079', '1167']
```

appMass2name_list (*mass*, *decimal_places=2*)
Creates a list of unimod names for a given approximate mass

Parameters **mass** (*float*) –

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Unimod names

Return type list

Examples::

```
>>> import ursgal
>>> U = ursgal.UnimodMapper()
>>> U.appMass2name_list(18, decimal_places=0)
['Fluoro', 'Methyl:2H(3)13C(1)', 'Xle->Met', 'Glu->Phe', 'Pro->Asp']
```

composition2id_list (*composition*)

Converts unimod composition to unimod name list, since a given composition can map to mutiple entries in the XML.

Parameters **composition** (*dict*) –

Returns Unimod IDs

Return type list

composition2mass (*composition*)
Converts unimod composition to unimod monoisotopic mass,

Parameters **composition** (*float*) –

Returns monoisotopic mass

Return type float

composition2name_list (*composition*)

Converts unimod composition to unimod name list, since a given composition can map to mutiple entries in the XML.

Parameters **composition** (*dict*) –

Returns Unimod names

Return type list

id2composition (*unimod_id*)
Converts unimod id to unimod composition

Parameters **unimod_id** (*int*) –

Returns Unimod elemental composition

Return type dict

id2mass(*unimod_id*)
Converts unimodID to unimod mass

Parameters **unimod_id** (*int*) –

Returns Unimod mono isotopic mass

Return type float

id2name(*unimod_id*)
Converts unimodID to unimod name

Parameters **unimod_id** (*int*) –

Returns Unimod name

Return type str

mass2composition_list(*mass*)

Converts unimod mass to unimod element composition list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) –

Returns Unimod elemental compositions

Return type list

mass2id_list(*mass*)

Converts unimod mass to unimod name list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) –

Returns Unimod IDs

Return type list

mass2name_list(*mass*)

Converts unimod mass to unimod name list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) –

Returns Unimod names

Return type list

name2composition(*unimod_name*)
Converts unimod name to unimod composition

Parameters **unimod_name** (*str*) –

Returns Unimod elemental composition

Return type dict

name2id(*unimod_name*)
Converts unimod name to unimodID

Parameters **unimod_name** (*str*) –

Returns Unimod id

Return type int

name2mass(*unimod_name*)
Converts unimod name to unimod mono isotopic mass

Parameters **unimod_name** (*str*) –

Returns Unimod mono isotopic mass

Return type float

writeXML(*modification_dict*, *xmlFile=None*)
Writes a unimod-style userdefined_unimod.xml file in ursal/resources/platform_independent/arc_independent/ext

Parameters

- **modification_dict** (*dict*) – dictionary containing at least
- **'mass'** (*mass of the modification*) –
- **'name'** (*name of the modificaton*) –
- **'composition'** (*chmical composition of the modification as a Hill notation*) –

4.7 Ursgal Results

The main Ursgal result file format is a comma separated value file (CSV). This is a table format that can be easily viewed and further processed. In general, each row contains a PSM and each column contains distinct properties of that PSM. The number of columns can vary depending on the engines (unodes) that have been run, but the most unified columns and other common columns are briefly described here.

4.7.1 Raw data location

Location and file name of the file (.mgf or .mzML) that contains the MS information leading to the identification of the PSM

4.7.2 Spectrum ID

ID of the mass spectrum leading to the identification of the PSM

4.7.3 Spectrum Title

Information about the spectrum leading to the identification of the PSM in the format '<file_name>.<spectrum_id>.<spectrum_id>.<charge>'

4.7.4 Retention Time (s)

Retention time corresponding to the mass spectrum leading to the identification of the PSM

4.7.5 Sequence

Peptide sequence matched to the spectrum

4.7.6 Protein ID

Protein ID corresponding to the peptide sequence matched to the spectrum. If multiple Protein IDs correspond to the same peptide sequence, they are separated by '<|>'

4.7.7 Sequence Start

Start of the peptide sequence in the corresponding protein sequence. Counting starts at 1 (not 0). Multiple start points are separated by ';' and start points of multiple corresponding proteins are separated by '<|>'

4.7.8 Sequence Stop

Stop of the peptide sequence in the corresponding protein sequence. Multiple stop points are separated by ';' and stop points of multiple corresponding proteins are separated by '<|>'

4.7.9 Sequence Pre AA

Amino acid before the start of the peptide sequence in the corresponding protein sequence. If a pre aa does not exist (start of the protein), this is indicated by '-'. Multiple pre aa are separated by ';' and pre aa of multiple corresponding proteins are separated by '<|>'

4.7.10 Sequence Post AA

Amino acid after the end of the peptide sequence in the corresponding protein sequence. If a post aa does not exist (end of the protein), this is indicated by '-'. Multiple post aa are separated by ';' and post aa of multiple corresponding proteins are separated by '<|>'

4.7.11 Charge

Charge of the (precursor) ion leading to the identification of the PSM.

4.7.12 Exp m/z

Experimental m/z of the (precursor) ion leading to the identification of the PSM.

4.7.13 uCalc m/z

Theoretical m/z of the peptide (including modifications) calculated by Ursgal. This can vary from the 'Calc m/z' which is reported by the engine corresponding to the PSM.

4.7.14 uCalc mass

Theoretical mass of the peptide (including modifications) calculated by Ursgal.

4.7.15 Accuracy (ppm)

Accuracy of the PSM based on the 'uCalc m/z' and 'Exp m/z'

4.7.16 Modifications

Modifications of the peptide sequence given in the following format: '<modification_name>:<position>' The modification name is given as PSI-MS name, the position corresponds to the position of the modification in the peptide. Position 0 corresponds to N-terminal modifications, position peptide_length+1 corresponds to C-terminal modifications. Multiple modifications within the same peptide sequence are separated by ';'

4.7.17 Mass Difference

Mass Difference (between the matched peptide and the precursor mass) as reported by open modification search engines. Multiple modifications within the same peptide sequence are separated by ';'

4.7.18 Mass Difference Annotations

Annotations of mass differences as reported by engines processing mass differences from open modification search engines. Multiple annotations within the same peptide sequence are separated by ';'

4.7.19 Complies search criteria

True or False, indicating if any conflicts with specified search parameters were identified. Conflicts are repoted in the column 'Conflicting uparam'

4.7.20 Enzyme Specificity

Indicated whether the matched peptide sequence originated from full enzymatic cleavage, semienzymatic cleave or nonspecific cleavage.

4.7.21 Is decoy

True or False, indicating whether the protein sequence corresponding to the PSM is a decoy sequence.

4.7.22 PEP

Posterior error probability of the PSM as determined by engines performing statistical post-processing.

4.7.23 q-value

q-value of the PSM as determined by engines performing statistical post-processing.

4.7.24 combined PEP

Combined PEP reported by the combined PEP approach used to combine results from multiple search engines. The basis for the combined PEP is the Bayes PEP which is calculated from the PEPs of each PSM reported by individual search engines.

4.7.25 Search Engine

Indicates which engine(s) reported the PSM.

CHAPTER 5

Parameters

5.1 Ursgal Parameters

A Dash script has been built in order to make it easier to explore uparams. It allows for searching for specific uparams, filtering by UNodes (i.e. engines), and filtering by utags. To install Dash, follow the instructions on

https://dash.plot.ly/

Afterwards, just go to the docs folder and execute the script:

```
user@localhost:~/ursgal/docs$ python uparams_to_dash.py
```

A local server will be created and you can view the interactive page:

http://127.0.0.1:8050/

Besides this, all uparams are still listed here as part of the documentation.

Note: This sphinx source file was **auto-generated** using ursgal/docs/source/conf.py, which parses ursgal/ursgal/uparams.py Please **do not** modify this file directly, but commit changes to ursgal.uparams.

5.1.1 -xmx

Set maximum Java heap size (used RAM)

Default value 13312m

type str

triggers rerun False

Available in unodes

- msgfplus2csv_v2016_09_16
- msgfplus2csv_v2017_01_27
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- ptmshepherd_0_3_5

Ursgal key translations for *-xmx*

Style	Translation
msfragger_style_1	-Xmx
msfragger_style_2	-Xmx
msfragger_style_3	-Xmx
msgfplus_style_1	-Xmx
mzidentml_style_1	-Xmx
pipi_style_1	-Xmx
ptmshepherd_style_1	-Xmx

5.1.2 aa_exception_dict

Unusual aminoacids that are not accepted (e.g. by unify_csv_1_0_0), but reported by some engines. Given as a dictionary mapping on he original_aa as well as the unimod modification name. U is now accepted as regular amino acid (2017/03/30).In Tag Graph this can be used to define amino acids other thanthe standard 20 to be included in the search.For those, chemical composition, monoisotopic mass and avg massas well as name and 3-letter code need to be given.

Default value ['J', 'O']

type dict

triggers rerun True

Available in unodes

- unify_csv_1_0_0
- upeptide_mapper_1_0_0
- compomics_utilities_4_11_5
- tag_graph_1_8_0

Ursgal key translations for *aa_exception_dict*

Style	Translation
compomics_utilities_style_1	aa_exception_dict
tag_graph_style_1	Amino Acids
unify_csv_style_1	aa_exception_dict
upeptide_mapper_style_1	aa_exception_dict

5.1.3 accept_conflicting_psms

If True, multiple PSMs for one spectrum can be reported if their score difference is below the threshold. If False, all PSMs for one spectrum are removed if the score difference between the best and secondbest PSM is not above the threshold, i.e. if there are conflicting PSMs with similar scores.

Default value False

type bool

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *accept_conflicting_psms*

Style	Translation
sanitize_csv_style_1	accept_conflicting_psms

5.1.4 add_contaminants

Contaminants are added automatically to the database by the search engine. PIPI uses the same contaminants database as MaxQuant

Default value False

type bool

triggers rerun True

Available in unodes

- pipi_1_4_5
- pipi_1_4_6

Ursgal key translations for *add_contaminants*

Style	Translation
pipi_style_1	add_contaminant

Ursgal value translations

Ursgal Value	Translated Value
.	pipi_style_1
0	0
1	1

5.1.5 allow_multiple_variable_mods_on_residue

Static mods are not considered

Default value False

type bool

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *allow_multiple_variable_mods_on_residue*

Style	Translation
msfragger_style_1	allow_multiple_variable_mods_on_residue
msfragger_style_2	allow_multiple_variable_mods_on_residue
msfragger_style_3	allow_multiple_variable_mods_on_residue

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
0	0	0	0
1	1	1	1

5.1.6 base_mz

m/z value that is used as basis for the conversion from ppm to Da

Default value 1000

type int

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- pipi_1_4_5
- pipi_1_4_6
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- deepnovo_0_0_1
- deepnovo_pointnovo
- tag_graph_1_8_0
- glycopeptide_fragmentor_1_0_0
- ptmshepherd_0_3_5

Ursgal key translations for *base_mz*

Style	Translation
deepnovo_style_1	base_mz
glycopeptide_fragmentor_style_1	base_mz
moda_style_1	base_mz
novor_style_1	base_mz
omssa_style_1	base_mz
pepnovo_style_1	base_mz
pipi_style_1	base_mz
ptmshepherd_style_1	base_mz
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
tag_graph_style_1	base_mz

5.1.7 batch_size

Sets the number of sequences loaded in as a batch from the database file

Default value 100000

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665

Ursgal key translations for *batch_size*

Style	Translation
msamanda_style_1	LoadedProteinsAtOnce
myrimatch_style_1	NumBatches
xtandem_style_1	spectrum, sequence batch size

5.1.8 batch_size_spectra

sets the number of spectra loaded into memory as a batch

Default value 2000

type int

triggers rerun False

Available in unodes

- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- deepnovo_0_0_1

Ursgal key translations for *batch_size_spectra*

Style	Translation
deepnovo_style_1	buffer_size
msamanda_style_1	LoadedSpectraAtOnce

5.1.9 bayesian_fold_change

Perform Bayesian protein fold change analysis

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *bayesian_fold_change*

Style	Translation
flash_lfq_style_1	–bay

5.1.10 bayesian_fold_change_control_condition

Control condition for bayesian fold change analysis

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *bayesian_fold_change_control_condition*

Style	Translation
flash_lfq_style_1	–ctr

5.1.11 bigger_scores_better

Defines if bigger scores are better (or the other way round), for scores that should be validated (see validation_score_field) e.g. by percolator, qvality

Default value None

type select

triggers rerun True

Available in unodes

- add_estimated_fdr_1_0_0
- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0
- qvality_2_02
- sanitize_csv_1_0_0
- svm_1_0_0
- ptminer_1_0

Ursgal key translations for *bigger_scores_better*

Style	Translation
add_estimated_fdr_style_1	bigger_scores_better
percolator_style_1	bigger_scores_better
ptminer_style_1	bigger_scores_better
qvality_style_1	-r
sanitize_csv_style_1	bigger_scores_better
svm_style_1	bigger_scores_better

Ursgal value translations

Ursgal Value	Translated Value				
.	add_estimated_fdr_style_1	percolator_style_1	qvality_style_1	sanitize_csv_style_1	s
None	None	None	None	None	N
deepnovo_0_0_1	1	1	1	1	1
deepnovo_pointnovo	1	1	1	1	1
mascot_x_x_x	1	1	1	1	1
moda_v1_51	1	1	1	1	1
moda_v1_61	1	1	1	1	1
moda_v1_62	1	1	1	1	1
msamanda_1_0_0_5242	1	1	1	1	1
msamanda_1_0_0_5243	1	1	1	1	1
msamanda_1_0_0_6299	1	1	1	1	1
msamanda_1_0_0_6300	1	1	1	1	1
msamanda_1_0_0_7503	1	1	1	1	1
msamanda_1_0_0_7504	1	1	1	1	1
msamanda_2_0_0_10695	1	1	1	1	1
msamanda_2_0_0_11219	1	1	1	1	1
msamanda_2_0_0_13723	1	1	1	1	1
msamanda_2_0_0_14665	1	1	1	1	1
msamanda_2_0_0_9695	1	1	1	1	1
msamanda_2_0_0_9706	1	1	1	1	1
msfragger_20170103	1	1	1	1	1
msfragger_20171106	1	1	1	1	1
msfragger_20190222	1	1	1	1	1
msfragger_20190628	1	1	1	1	1
msfragger_2_3	1	1	1	1	1
msfragger_3_0	1	1	1	1	1
msgfplus_v2016_09_16	0	0	0	0	0
msgfplus_v2017_01_27	0	0	0	0	0
msgfplus_v2018_01_30	0	0	0	0	0
msgfplus_v2018_06_28	0	0	0	0	0
msgfplus_v2018_09_12	0	0	0	0	0
msgfplus_v2019_01_22	0	0	0	0	0
msgfplus_v2019_04_18	0	0	0	0	0
msgfplus_v2019_07_03	0	0	0	0	0
msgfplus_v9979	0	0	0	0	0
myrimatch_2_1_138	1	1	1	1	1

Continued on

Table 1 – continued from previous page

Ursgal Value	Translated Value				
.	add_estimated_fdr_style_1	percolator_style_1	qvality_style_1	sanitize_csv_style_1	s
myrimatch_2_2_140	1	1	1	1	1
omssa_2_1_9	0	0	0	0	0
pglyco_db_2_2_0	1	1	1	1	1
pglyco_db_2_2_2	1	1	1	1	n/
pipi_1_4_5	1	1	1	1	1
pipi_1_4_6	1	1	1	1	1
pnovo_3_1_3	1	1	1	1	n/
tag_graph_1_8_0	1	1	1	1	1
xtandem_alanine	1	1	1	1	1
xtandem_cyclone_2010	1	1	1	1	1
xtandem_jackhammer	1	1	1	1	1
xtandem_piledriver	1	1	1	1	1
xtandem_sledgehammer	1	1	1	1	1
xtandem_vengeance	1	1	1	1	1

5.1.12 build_pyqms_result_index

Build index for faster access

Default value True

type bool

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *build_pyqms_result_index*

Style	Translation
pyqms_style_1	BUILD_RESULT_INDEX
sugarpy_plot_style_1	BUILD_RESULT_INDEX
sugarpy_run_style_1	BUILD_RESULT_INDEX

5.1.13 calibrate_mass

Perform mass calibration

Default value False

type bool

triggers rerun True

Available in unodes

- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *calibrate_mass*

Style	Translation
msfragger_style_2	calibrate_mass
msfragger_style_3	calibrate_mass

Ursgal value translations

Ursgal Value	Translated Value	
.	msfragger_style_2	msfragger_style_3
0	0	0
1	1	1

5.1.14 cleavage_cterm_mass_change

The mass added to the peptide C-terminus by protein cleavage

Default value 17.00305

type float

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *cleavage_cterm_mass_change*

Style	Translation
xtandem_style_1	protein, cleavage C-terminal mass change

5.1.15 cleavage_nterm_mass_change

The mass added to the peptide N-terminus by protein cleavage

Default value 1.00794

type float

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *cleavage_nterm_mass_change*

Style	Translation
xtandem_style_1	protein, cleavage N-terminal mass change

5.1.16 clip_nterm_m

Specifies the trimming of a protein N-terminal methionine as a variable modification

Default value False

type bool

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *clip_nterm_m*

Style	Translation
msfragger_style_1	clip_nTerm_M
msfragger_style_2	clip_nTerm_M
msfragger_style_3	clip_nTerm_M

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
0	0	0	0
1	1	1	1

5.1.17 compensate_small_fasta

Compensate for very small database files.

Default value False

type bool

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *compensate_small_fasta*

Style	Translation
xtandem_style_1	scoring, cyclic permutation

Ursgal value translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

5.1.18 compomics_utility_name

Default value accesses the PeptideMapper tool, other tools are not implemented/covered yet

Default value com.compomics.util.experiment.identification.protein_inference.executable.PeptideMapping

type str

triggers rerun True

Available in unodes

- compomics_utilities_4_11_5

Ursgal key translations for *compomics_utility_name*

Style	Translation
compomics_utilities_style_1	compomics_utility_name

5.1.19 compomics_version

Defines the compomics version to use

Default value compomics_utilities_4_11_5

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *compomics_version*

Style	Translation
ucontroller_style_1	compomics_version

5.1.20 compress_raw_search_results_if_possible

Compress raw search result to .gz: True or False

Default value True

type bool

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *compress_raw_search_results_if_possible*

Style	Translation
ucontroller_style_1	compress_raw_search_results_if_possible

Ursgal value translations

Ursgal Value	Translated Value
.	ucontroller_style_1
crux_2_1	0
deepnovo_0_0_1	0
deepnovo_pointnovo	0
kojak_1_5_3	0
mascot_x_x_x	1
moda_v1_51	0
moda_v1_61	0
moda_v1_62	0
msamanda_1_0_0_5242	0
msamanda_1_0_0_5243	0
msamanda_1_0_0_6299	0
msamanda_1_0_0_6300	0
msamanda_1_0_0_7503	0
msamanda_1_0_0_7504	0
msamanda_2_0_0_10695	0
msamanda_2_0_0_11219	0
msamanda_2_0_0_13723	0
msamanda_2_0_0_14665	0
msamanda_2_0_0_9695	0
msamanda_2_0_0_9706	0
msfragger_20170103	0
msfragger_20171106	0
msfragger_20190222	0
msfragger_20190628	0
msfragger_2_3	0
msfragger_3_0	0
msgfplus_v2016_09_16	1
msgfplus_v2017_01_27	1
msgfplus_v2018_01_30	1
msgfplus_v2018_06_28	1
msgfplus_v2018_09_12	1
msgfplus_v2019_01_22	1
msgfplus_v2019_04_18	1
msgfplus_v2019_07_03	1
msgfplus_v9979	1
myrimatch_2_1_138	1
myrimatch_2_2_140	1
novor_1_05	0
novor_1_1beta	0
omssa_2_1_9	0

Continued on next page

Table 2 – continued from previous page

Ursgal Value	Translated Value
.	ucontroller_style_1
pepnovo_3_1	0
pglyco_db_2_2_0	0
pglyco_db_2_2_2	0
pipi_1_4_5	0
pipi_1_4_6	0
xtandem_alanine	1
xtandem_cyclone_2010	1
xtandem_jackhammer	1
xtandem_piledriver	1
xtandem_sledgehammer	1
xtandem_vengeance	1

5.1.21 compute_xcorr

Compute xcorr

Default value False

type bool

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *compute_xcorr*

Style	Translation
myrimatch_style_1	ComputeXCorr

Ursgal value translations

Ursgal Value	Translated Value
.	myrimatch_style_1
0	0
1	1

5.1.22 consecutive_ion_prob

Probability of consecutive ion (used in correlation correction)

Default value 0.5

type float

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *consecutive_ion_prob*

Style	Translation
omssa_style_1	-scorp

5.1.23 convert_aa_in_motif

Convert a single aminoacid in a sequence motif into another characeter using a string "new_aa,motif,position_to_be_replaced" where new_aa is the new character, motif is the regular expression that identifies the sequenc motif and position_to_be_replaced is the position in the motif that should be replaced (e.g. use "J,N[^P][ST],0" to convert N-X-S/T into J-X-S/T

Default value None

type str

triggers rerun True

Available in unodes

- generate_target_decoy_1_0_0

Ursgal key translations for *convert_aa_in_motif*

Style	Translation
generate_target_decoy_style_1	convert_aa_in_motif

5.1.24 convert_to_sfinx

If True, the header of the identifier column is "rownames". If False, the joined identifier header name will be used

Default value False

type bool

triggers rerun True

Available in unodes

- csv2counted_results_1_0_0

Ursgal key translations for *convert_to_sfinx*

Style	Translation
csv2counted_results_style_1	convert2sfinx

5.1.25 count_by_file

the number of unique hits for each identifier is given in separate columns for each raw file (file name as defiened in Spectrum Title)

Default value True

type bool

triggers rerun True

Available in unodes

- csv2counted_results_1_0_0

Ursgal key translations for *count_by_file*

Style	Translation
csv2counted_results_style_1	count_by_file

5.1.26 count_column_names

List of column headers which are used for counting. The combination of these headers creates the unique countable element.

Default value [‘Sequence’, ‘Modifications’]

type list

triggers rerun True

Available in unodes

- csv2counted_results_1_0_0

Ursgal key translations for *count_column_names*

Style	Translation
csv2counted_results_style_1	count_column_names

5.1.27 cpus

Number of used cpus/threads

-1 : 'max - 1' >0 : cpu num

Default value -1

type int _uevaluation_req

triggers rerun False

Available in unodes

- kojak_1_5_3
- moda_v1_61
- moda_v1_62
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- ucontroller
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

- pipi_1_4_5
- pipi_1_4_6
- moda_v1_62
- moda_v1_61
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- pnovo_3_1_3
- flash_lfq_1_1_1
- ptmshepherd_0_3_5

Ursgal key translations for *cpus*

Style	Translation
flash_lfq_style_1	–thr
kojak_style_1	cpus
moda_style_1	-@
msfragger_style_1	num_threads
msfragger_style_2	num_threads
msfragger_style_3	num_threads
msgfplus_style_1	-thread
myrimatch_style_1	-cpus
omssa_style_1	-nt
pglyco_db_style_1	process
pipi_style_1	thread_num
pnovo_style_1	thread
ptmshepherd_style_1	threads
ucontroller_style_1	cpus
xtandem_style_1	spectrum, threads

Ursgal value translations

Ursgal Value	Translated Value												
.	kojak_style_1	moda_style_1	msfragger_style_1	msfragger_style_2	msfragger_style_3	msgfplus_style_1	myrimatch_style_1	omssa_style_1	pglyco_db_style_1	pipi_style_1	pnovo_style_1	ucontroller_style_1	xtandem_style_1
-1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1

5.1.28 cross_link_definition

Cross-link and mono-link masses allowed. May have more than one of each parameter. Format for cross_link is:

[amino acids] [amino acids] [mass mod] [identifier]

One or more amino acids (uppercase only!!) can be specified for each linkage moiety. Use lowercase 'n' or 'c' to indicate protein N-terminus or C-terminus

Default value nK nK 138.0680742 BS3

type str

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *cross_link_definition*

Style	Translation
kojak_style_1	cross_link_definition

5.1.29 csv_filter_rules

Rules are defined as list of lists with three elements:

1. the column name/csv fieldname,
2. the rule,
3. the value which should be compared

e.g.: ['Is decoy', 'equals', 'false']

Default value None

type list

triggers rerun True

Available in unodes

- filter_csv_1_0_0

Ursgal key translations for *csv_filter_rules*

Style	Translation
filter_csv_style_1	filter_rules

Ursgal value translations

Ursgal Value	Translated Value
.	filter_csv_style_1

5.1.30 database

Path to database file containing protein sequences in fasta format.

Default value None

type str

triggers rerun True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- unify_csv_1_0_0

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- upeptide_mapper_1_0_0
- compomics_utilities_4_11_5
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- deepnovo_0_0_1
- tag_graph_1_8_0
- ptminer_1_0

Ursgal key translations for *database*

Style	Translation
compomics_utilities_style_1	database
deepnovo_style_1	db_fasta_file
kojak_style_1	database
moda_style_1	Fasta
msamanda_style_1	database
msfragger_style_1	database_name
msfragger_style_2	database_name
msfragger_style_3	database_name
msgfplus_style_1	-d
myrimatch_style_1	ProteinDatabase
omssa_style_1	-d
pglyco_db_style_1	fasta
pipi_style_1	db
ptminer_style_1	protein_database
tag_graph_style_1	fmindex
unify_csv_style_1	database
upeptide_mapper_style_1	database
xtandem_style_1	file URL

5.1.31 database_taxonomy

If a taxonomy ID is specified, only the corresponding protein sequences from the fasta database are included in the search.

Default value all

type str

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *database_taxonomy*

Style	Translation
xtandem_style_1	taxon label

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
all	0

5.1.32 de_novo_results

Path to the unified de novo results used as input for TagGraph

Default value None

type str

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *de_novo_results*

Style	Translation
tag_graph_style_1	de_novo

5.1.33 decoy_generation_mode

Decoy database: creates a target decoy database based on shuffling of peptides (shuffle_peptide) or complete reversing the protein sequence (reverse_protein).

Default value shuffle_peptide

type select

triggers rerun True

Available in unodes

- generate_target_decoy_1_0_0

Ursgal key translations for *decoy_generation_mode*

Style	Translation
generate_target_decoy_style_1	mode

5.1.34 decoy_tag

decoy-specific tag to differentiate between targets and decoys

Default value decoy_

type str

triggers rerun True

Available in unodes

- generate_target_decoy_1_0_0
- kojak_1_5_3
- myrimatch_2_1_138
- myrimatch_2_2_140
- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7
- unify_csv_1_0_0
- xtandem2csv_1_0_0
- upeptide_mapper_1_0_0
- percolator_3_2_1
- ptminer_1_0
- percolator_3_4_0
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *decoy_tag*

Style	Translation
generate_target_decoy_style_1	decoy_tag
kojak_style_1	decoy_tag
msfragger_style_2	decoy_prefix
msfragger_style_3	decoy_prefix
myrimatch_style_1	DecoyPrefix
mzidentml_style_1	-decoyRegex
percolator_style_1	-P
ptminer_style_1	decoy_tag
unify_csv_style_1	decoy_tag
upeptide_mapper_style_1	decoy_tag
xtandem2csv_style_1	decoy_tag

5.1.35 deepnovo_beam_search

DeepNovo builds beam search

Default value True

type bool

triggers rerun True

Available in unodes

- deepnovo_0_0_1

Ursgal key translations for *deepnovo_beam_search*

Style	Translation
deepnovo_style_1	beam_search

5.1.36 deepnovo_beam_size

Number of optimal paths to search during decoding

Default value 5

type int

triggers rerun True

Available in unodes

- deepnovo_0_0_1
- deepnovo_pointnovo

Ursgal key translations for *deepnovo_beam_size*

Style	Translation
deepnovo_style_1	beam_size

5.1.37 deepnovo_build_knapsack

DeepNovo builds the knapsack matrix

Default value False

type bool

triggers rerun True

Available in unodes

- deepnovo_0_0_1

Ursgal key translations for *deepnovo_build_knapsack*

Style	Translation
deepnovo_style_1	knapsack_build

5.1.38 deepnovo_direction

Defines the direction for DeepNovo

Default value bi_directional

type select

triggers rerun True

Available in unodes

- deepnovo_0_0_1

Ursgal key translations for *deepnovo_direction*

Style	Translation
deepnovo_style_1	direction

Ursgal value translations

Ursgal Value	Translated Value
.	deepnovo_style_1
bi_directional	2
forward	0
reverse	1

5.1.39 deepnovo_knapsack_file

Path to the knapsack matrix for DeepNovo. Use “default” for the default file location in the resources

Default value default

type str

triggers rerun True

Available in unodes

- deepnovo_0_0_1
- deepnovo_pointnovo

Ursgal key translations for *deepnovo_knapsack_file*

Style	Translation
deepnovo_style_1	knapsack_file

5.1.40 deepnovo_mode

Defines the search mode for DeepNovo

Default value search_denovo

type select

triggers rerun True

Available in unodes

- deepnovo_0_0_1
- deepnovo_pointnovo

Ursgal key translations for *deepnovo_mode*

Style	Translation
deepnovo_style_1	('search_denovo', 'search_hybrid', 'search_db', 'decode')

5.1.41 deepnovo_shared_weights

DeepNovo uses shared weights

Default value True

type bool

triggers rerun True

Available in unodes

- deepnovo_0_0_1

Ursgal key translations for *deepnovo_shared_weights*

Style	Translation
deepnovo_style_1	shared

5.1.42 deepnovo_use_intensity

DeepNovo uses intensity

Default value True

type bool

triggers rerun True

Available in unodes

- deepnovo_0_0_1

Ursgal key translations for *deepnovo_use_intensity*

Style	Translation
deepnovo_style_1	use_intensity

5.1.43 deepnovo_use_lstm

DeepNovo uses lstm

Default value False

type bool

triggers rerun True

Available in unodes

- deepnovo_0_0_1
- deepnovo_pointnovo

Ursgal key translations for *deepnovo_use_lstm*

Style	Translation
deepnovo_style_1	use_lstm

5.1.44 deisotope_spec

Perform Deisotoping for MS2 spectra. Options are: “none”, “deisotope_with_singleton_charge_one”, “perform_deisotoping”

Default value deisotope_with_singleton_charge_one

type select

triggers rerun True

Available in unodes

- msamanda_2_0_0_9695
- msamanda_2_0_0_9706
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msfragger_3_0

Ursgal key translations for *deisotope_spec*

Style	Translation
msamanda_style_1	PerformDeisotoping
msfragger_style_3	deisotope

Ursgal value translations

Ursgal Value	Translated Value	
.	msamanda_style_1	msfragger_style_3
deisotope_with_singleton_charge_one	n/t	1
none	false	0
perform_deisotoping	true	2

5.1.45 del_from_params_before_json_dump

List of parameters that are deleted before .json is dumped (to not overload the .json with unimportant informations)

Default value ['grouped_psms']

type list

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *del_from_params_before_json_dump*

Style	Translation
ucontroller_style_1	del_from_params_before_json_dump

5.1.46 delta_mass_exclude_range

The given mass range is excluded from searching for shifted ions.

Default value [0.0, 0.0]

type list

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *delta_mass_exclude_range*

Style	Translation
msfragger_style_3	delta_mass_exclude_ranges

5.1.47 deltamass_allowed_residues

Only used in MSFragger labile mode, specifies which amino acids are allowed to contain a labile modification

Default value

type str

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *deltamass_allowed_residues*

Style	Translation
msfragger_style_3	deltamass_allowed_residues

Ursgal value translations

Ursgal Value	Translated Value
.	msfragger_style_3

5.1.48 denovo_model

PepNovo model used for de novo sequencing. Based on the enzyme and fragmentation type. Currently only CID_IT_TRYP available.

Default value cid_trypsin

type select

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *denovo_model*

Style	Translation
pepnovo_style_1	-model

Ursgal value translations

Ursgal Value	Translated Value
.	pepnovo_style_1
cid_trypsin	CID_IT_TRYP

5.1.49 denovo_model_dir

Directory containing the model files de novo sequencing. Use "default" for the default folder of the engine (DeepNovo: <deepnovo_resources>/train.example; PepNovo: resources/<platform>/<architecture>/pepnovo_3_1)

Default value default

type str

triggers rerun True

Available in unodes

- pepnovo_3_1
- deepnovo_0_0_1
- deepnovo_pointnovo

Ursgal key translations for *denovo_model_dir*

Style	Translation
deepnovo_style_1	train_dir
pepnovo_style_1	-model_dir

Ursgal value translations

Ursgal Value	Translated Value
.	pepnovo_style_1
default	None

5.1.50 determine_localization

specifying whether positions for mass shifts should be determined or not

Default value True

type bool

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *determine_localization*

Style	Translation
ptminer_style_1	is_localized

Ursgal value translations

Ursgal Value	Translated Value
.	ptminer_style_1
0	0
1	1

5.1.51 determine_unimod_annotation

specifying whether mass shifts should be annotated or not

Default value True

type bool

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *determine_unimod_annotation*

Style	Translation
ptminer_style_1	is_annotated

Ursgal value translations

Ursgal Value	Translated Value
.	ptminer_style_1
0	0
1	1

5.1.52 diagnostic_fragments

Specify molecules (or masses) that will be used as diagnostic fragments in the search. Assuming a charge of 1, the mass of a proton is added to all molecules (and masses). Specify as a dictionary with the keys "chemical_formulas", "unimods", "glycans", "masses", and lists with the corresponding molecules as values.

Default value ['chemical_formulas', 'glycans', 'masses', 'unimods']

type dict

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *diagnostic_fragments*

Style	Translation
msfragger_style_3	diagnostic_fragments

5.1.53 engine_internal_decoy_generation

Engine creates an own decoy database. Not recommended, because a target decoy database should be generated independently from the search engine, e.g. by using the uNode generate_target_decoy_1_0_0

Default value False

type bool

triggers rerun True

Available in unodes

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243

- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- pipi_1_4_5
- pipi_1_4_6

Ursgal key translations for *engine_internal_decoy_generation*

Style	Translation
msamanda_style_1	generate_decoy
msgfplus_style_1	-tda
pipi_style_1	add_decoy
xtandem_style_1	scoring, include reverse

Ursgal value translations

Ursgal Value	Translated Value			
.	msamanda_style_1	msgfplus_style_1	pipi_style_1	xtandem_style_1
0	false	0	0	no
1	true	1	1	yes

5.1.54 engines_create_folders

Create folders for the output of engines that allow this option in their META_INFO ('create_own_folder' : True). True or False

Default value True

type bool

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *engines_create_folders*

Style	Translation
ucontroller_style_1	engines_create_folders

5.1.55 enzyme

Enzyme: Rule of protein cleavagePossible cleavages are : argc -> [R]|{P} aspn -> [X]|[D] aspn_gluc chymotrypsin -> [FMWY]|{P} chymotrypsin_p -> [FMWY]|[X] cnbr -> [M]|{P} elastase -> [AGILV]|{P} formic_acid -> [D]|{P} gluc lysc lysc_p lysn no_cleavage nonspecific pepsina semi_chymotrypsin semi_gluc semi_tryptic thermolysin_p top_down trypsin trypsin_chymotrypsin trypsin_cnbr trypsin_p lysc_gluc

Default value trypsin

type select

triggers rerun True

Available in unodes

- generate_target_decoy_1_0_0
- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242

- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0
- unify_csv_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- deepnovo_0_0_1
- deepnovo_pointnovo
- pnovo_3_1_3
- tag_graph_1_8_0

Ursgal key translations for *enzyme*

Style	Translation
deepnovo_style_1	cleavage_rule
generate_target_decoy_style_1	enzyme
kojak_style_1	enzyme
moda_style_1	Enzyme
msamanda_style_1	enzyme specificity
msfragger_style_1	enzyme
msfragger_style_2	enzyme
msfragger_style_3	enzyme
msgfplus_style_1	-e
myrimatch_style_1	CleavageRules
novor_style_1	enzyme
omssa_style_1	-e
pepnovo_style_1	-digest
percolator_style_1	enz
pglyco_db_style_1	enzyme
pipi_style_1	enzyme
pnovo_style_1	enzyme
tag_graph_style_1	Enzyme
unify_csv_style_1	enzyme
xtandem_style_1	protein, cleavage site

Ursgal value translations

Ursgal Value				
.	deepnovo_style_1	generate_target_decoy_style_1	kojak_style_1	moda_style_1
alpha_lp	n/t	n/t	n/t	n/t
argc	arg-c	R;C;P	n/t	argc, R/C
aspn	asp-n	D;N;	n/t	aspn, D/N;
aspn_gluc	n/t	n/t	n/t	n/t
chymotrypsin	n/t	FMWY;C;P	n/t	chymotrypsin, FMWY/C
chymotrypsin_p	n/t	FMWY;C;	n/t	chymotrypsin, FMWY/C
clostripain	clostripain	R;C;	n/t	clostripain, R/C
cnbr	cnbr	M;C;P	n/t	cnbr, M/C
elastase	n/t	AGILV;C;P	n/t	elastase, AGILV/C
formic_acid	formic acid	D;C;P	n/t	formic_acid, D/C
formic_acid_p	n/t	n/t	n/t	n/t
gluc	n/t	DE;C;P	[DE]|{P}	gluc, DE/C
gluc_bicarb	n/t	E;C;P	n/t	gluc_bicarb, E/C
iodosobenzoate	n/t	W;C;	n/t	iodosobenzoate, W/C
lysc	lysc	K;C;P	n/t	lysc, K/C
lysc_gluc	n/t	DEK;C;P	[DEK]|{P}	n/t
lysc_p	n/t	K;C;	n/t	lysc_p, K/C
lysn	n/t	K;N;	|[K]	lysn, K/N
lysn_promisc	n/t	AKRS;N;	n/t	lysn_promisc, AKRS/N
no_cleavage	n/t	n/t	n/t	NONE
nonspecific	n/t	n/t	n/t	n/t
pepsina	n/t	FL;C;	n/t	pepsina, FL/C
protein_endopeptidase	n/t	P;C;	n/t	protein_endopeptidase, P/C
staph_protease	n/t	E;C;	n/t	staph_protease, E/C
tca	n/t	n/t	n/t	n/t
thermolysin_p	n/t	n/t	n/t	n/t
top_down	n/t	n/t	n/t	n/t
trypsin	trypsin	KR;C;P	[KR]|{P}	trypsin, KR/C
trypsin_chymotrypsin	n/t	n/t	n/t	n/t
trypsin_cnbr	n/t	KRM;C;P	n/t	trypsin_cnbr, KRM/C
trypsin_gluc	n/t	DEKR;C;P	n/t	trypsin_gluc, DEKR/C
trypsin_p	n/t	KR;C;	[RK]|	trypsin_p, KR/C

5.1.56 evidence_score_field

Field which is used for scoring in pyqms_1_0_0

Default value PEP

type str

triggers rerun True

Available in unodes

- pyqms_1_0_0

Ursgal key translations for *evidence_score_field*

Style	Translation
pyqms_style_1	evidence_score_field

5.1.57 experiment_setup

Default value []

type list

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *experiment_setup*

flash_lfq_style_1 experiment_setup

Format: {“1”: {“FileName”: <filename as in identfile>, “Condition”:<str>, “Biorep”: <int>, “Fraction”: <int>, “Techrep”: <int>}, “2”: .... }

5.1.58 extract_venndiagram_file

The user can retrieve a csv file containing results from the venn diagram

Default value False

type bool

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *extract_venndiagram_file*

Style	Translation
venndiagram_style_1	extract_venndiagram_file

5.1.59 fdr_cutoff

Target PSMs with a lower FDR than this threshold will be used as a positive training set for SVM post-processing

Default value 0.01

type float

triggers rerun True

Available in unodes

- svm_1_0_0

Ursgal key translations for *fdr_cutoff*

Style	Translation
svm_style_1	fdr_cutoff

5.1.60 fdr_method

specifying the fdr method to use

Default value transferred

type str

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *fdr_method*

Style	Translation
ptminer_style_1	fdr_method

Ursgal value translations

Ursgal Value	Translated Value
.	ptminer_style_1
global	1
separate	2
transferred	3

5.1.61 fixed_label_isotope_enrichment_levels

Enrichment of labeled elements in labeled chemical used

Default value ['13C', '15N', '2H']

type dict

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *fixed_label_isotope_enrichment_levels*

Style	Translation
pyqms_style_1	FIXED_LABEL_ISOTOPE_ENRICHMENT_LEVELS
sugarpy_plot_style_1	FIXED_LABEL_ISOTOPE_ENRICHMENT_LEVELS
sugarpy_run_style_1	FIXED_LABEL_ISOTOPE_ENRICHMENT_LEVELS

5.1.62 fold_change_cutoff

fold-change cutoff for Bayesian protein fold-change analysis

Default value 0.1

type float

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *fold_change_cutoff*

Style	Translation
flash_lfq_style_1	–fcc

5.1.63 forbidden_cterm_mods

List of modifications (unimod name) that are not allowed to occur at the C-terminus of a peptide, e.g. ['GG']

Default value []

type list

triggers rerun True

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *forbidden_cterm_mods*

Style	Translation
xtandem_style_1	residue, potential modification mass

5.1.64 forbidden_residues

Aminoacids that are not allowed during/taken into account during denovo searches. Given as a string of comma seperated aminoacids (single letter code)

Default value I,U

type str

triggers rerun True

Available in unodes

- novor_1_1beta
- novor_1_05

Ursgal key translations for *forbidden_residues*

Style	Translation
novor_style_1	forbiddenResidues

5.1.65 force

If set 'True', engines are forced to re-run although no node-related parameters have changed

Default value False

type bool

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *force*

Style	Translation
ucontroller_style_1	force

5.1.66 frag_clear_mz_range

Removes peaks in this m/z range prior to matching. Given as list [min_clear_mz, max_clear_mz]. Useful for iTRAQ/TMT experiments, i.e. [0.0, 150.0].

Default value [0.0, 0.0]

type list

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6

Ursgal key translations for *frag_clear_mz_range*

Style	Translation
msfragger_style_1	clear_mz_range
msfragger_style_2	clear_mz_range
msfragger_style_3	clear_mz_range
pipi_style_1	frag_clear_mz_range

5.1.67 frag_mass_tolerance

Mass tolerance of measured and calculated fragment ions

Default value 20

type int

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- pnovo_3_1_3
- tag_graph_1_8_0
- deepnovo_pointnovo
- glycopeptide_fragmentor_1_0_0
- ptmshepherd_0_3_5

Ursgal key translations for *frag_mass_tolerance*

Style	Translation
deepnovo_style_1	AA_MATCH_PRECISION
glycopeptide_fragmentor_style_1	frag_mass_tolerance
moda_style_1	FragTolerance
msamanda_style_1	ms2_tol
msfragger_style_1	fragment_mass_tolerance
msfragger_style_2	fragment_mass_tolerance
msfragger_style_3	fragment_mass_tolerance
myrimatch_style_1	FragmentMzTolerance
novor_style_1	fragmentIonErrorTol
omssa_style_1	-to
pepnovo_style_1	-fragment_tolerance
pglyco_db_style_1	search_fragment_tolerance
pipi_style_1	ms2_tolerance
pnovo_style_1	frag_tol
ptminer_style_1	fragment_tol
ptmshepherd_style_1	spectra_ppmtol
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
tag_graph_style_1	ppmstd
xtandem_style_1	spectrum, fragment monoisotopic mass error

5.1.68 frag_mass_tolerance_unit

Fragment mass tolerance unit: available in ppm (parts-per-millon), da (Dalton) or mmu (Milli mass unit)

Default value ppm

type select

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- pnovo_3_1_3
- tag_graph_1_8_0
- deepnovo_pointnovo
- glycopeptide_fragmentor_1_0_0
- ptmshepherd_0_3_5

Ursgal key translations for *frag_mass_tolerance_unit*

Style	Translation
deepnovo_style_1	AA_MATCH_PRECISION
glycopeptide_fragmentor_style_1	frag_mass_tolerance_unit
moda_style_1	FragTolerance
msamanda_style_1	ms2_tol unit
msfragger_style_1	fragment_mass_units
msfragger_style_2	fragment_mass_units
msfragger_style_3	fragment_mass_units
myrimatch_style_1	FragmentMzTolerance
novor_style_1	fragmentIonErrorTol
omssa_style_1	frag_mass_tolerance_unit
pepnovo_style_1	frag_mass_tolerance_unit
pglyco_db_style_1	search_fragment_tolerance_type
pipi_style_1	frag_mass_tolerance_unit
pnovo_style_1	frag_tol_type_ppm
ptminer_style_1	fragment_tol_type
ptmshepherd_style_1	spectra_ppmtol
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
tag_graph_style_1	frag_mass_tolerance_unit
xtandem_style_1	spectrum, fragment monoisotopic mass error units

Ursgal value translations

Ursgal Value	Translated Value										
.	msamanda	ms-fragger_style_1	ms-fragger_style_1	ms-fragger_style_2	myrimatch_style_1	novor_style_1	omssa_style_1	pglyco_db_style_1	pnovo_style_1	prosminer_style_1	xtandem_style_1
da	Da	0	0	0	Da	Da	Da	Da	0	0	Daltons
ppm	n/t	1	1	1	n/t	n/t	n/t	n/t	1	1	n/t

5.1.69 frag_mass_type

Fragment mass type: monoisotopic or average

Default value monoisotopic

type select

triggers rerun True

Available in unodes

- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *frag_mass_type*

Style	Translation
omssa_style_1	-tom
xtandem_style_1	spectrum, fragment mass type

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
average	1
monoisotopic	0

5.1.70 frag_max_charge

Maximum fragment ion charge to search.

Default value 4

type int

triggers rerun True

Available in unodes

- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *frag_max_charge*

Style	Translation
msfragger_style_1	max_fragment_charge
msfragger_style_2	max_fragment_charge
msfragger_style_3	max_fragment_charge
omssa_style_1	-zoh

5.1.71 frag_method

Used fragmentation method, e.g. collision-induced dissociation (cid), electron-capture dissociation (ecd), electron-transfer dissociation (etd), Higher-energy C-trap dissociation (hcd)

Default value hcd

type select

triggers rerun True

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22

- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- novor_1_1beta
- novor_1_05
- pnovo_3_1_3

Ursgal key translations for *frag_method*

Style	Translation
msgfplus_style_1	-m
novor_style_1	fragmentation
pnovo_style_1	activation_type

Ursgal value translations

Ursgal Value	Translated Value		
.	msgfplus_style_1	novor_style_1	pnovo_style_1
cid	1	CID	CID
etd	2	n/t	ETD
hcd	3	HCD	HCD

5.1.72 frag_min_mz

Minimal considered fragment ion m/z

Default value 150

type int

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *frag_min_mz*

Style	Translation
xtandem_style_1	spectrum, minimum fragment mz

5.1.73 ftp_blocksize

Blocksize for ftp download

Default value 1024

type int

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_blocksize*

Style	Translation
get_ftp_style_1	ftp_blocksize

5.1.74 ftp_folder

ftp folder that should be downloaded

Default value None

type str

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_folder*

Style	Translation
get_ftp_style_1	ftp_folder

5.1.75 ftp_include_ext

Only files with the defined file extension are downloaded with ftp download

Default value None

type str

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_include_ext*

Style	Translation
get_ftp_style_1	ftp_include_ext

5.1.76 ftp_login

Login name/user for the ftp server e.g. "PASS00269" in peptideatlas.orgftp download '' : None

Default value None

type str

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_login*

Style	Translation
get_ftp_style_1	ftp_login

5.1.77 ftp_max_number_of_files

Maximum number of files that will be downloaded 0 : No Limitation

Default value None

type int

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_max_number_of_files*

Style	Translation
get_ftp_style_1	ftp_max_number_of_files

5.1.78 ftp_output_folder

Default ftp download path '' : None

Default value None

type str

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_output_folder*

Style	Translation
get_ftp_style_1	ftp_output_folder

5.1.79 ftp_password

ftp download password '' : None

Default value None

type str_password

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_password*

Style	Translation
get_ftp_style_1	ftp_password

5.1.80 ftp_url

ftp download URL, will fail if it is not set by the user '' : None

Default value None

type str

triggers rerun True

Available in unodes

- get_ftp_files_1_0_0

Ursgal key translations for *ftp_url*

Style	Translation
get_ftp_style_1	ftp_url

5.1.81 glycans_incl_as_mods

List of Unimod PSI-MS names corresponding to glycans that were included in the database search as modification (will be removed from the peptidoform by SugarPy).

Default value ['HexNAc', 'HexNAc(2)']

type list

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *glycans_incl_as_mods*

Style	Translation
sugarpy_plot_style_1	unimod_glycans_incl_in_search
sugarpy_run_style_1	unimod_glycans_incl_in_search

5.1.82 header_translations

Translate output headers into Ursgal unify_csv style headers 'None' : None

Default value None

type str

triggers rerun True

Available in unodes

- kojak_percolator_2_08
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus2csv_v2016_09_16
- msgfplus2csv_v2017_01_27
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- msgfplus2csv_v2017_07_04
- msgfplus2csv_v1_2_0
- msgfplus2csv_v1_2_1
- pipi_1_4_5
- pipi_1_4_6
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- pglyco_fdr_2_2_0
- pglyco_fdr_2_2_2
- deepnovo_0_0_1

- deepnovo_pointnovo
- tag_graph_1_8_0
- flash_lfq_1_1_1

Ursgal key translations for *header_translations*

Style	Translation
deepnovo_style_1	header_translations
flash_lfq_style_1	header_translations
kojak_percolator_style_1	header_translations
msamanda_style_1	header_translations
msfragger_style_1	header_translations
msfragger_style_2	header_translations
msfragger_style_3	header_translations
msgfplus_style_1	header_translations
novor_style_1	header_translations
omssa_style_1	header_translations
pepnovo_style_1	header_translations
pglyco_db_style_1	header_translations
pglyco_fdr_style_1	header_translations
pipi_style_1	header_translations
tag_graph_style_1	header_translations

Ursgal value translations

Ursgal Value			
.	deepnovo_style_1	flash_lfq_style_1	koja
Accession	n/t	n/t	n/t
Charge	n/t	n/t	n/t
Defline	n/t	n/t	n/t
E-value	n/t	n/t	n/t
Filename/id	n/t	n/t	n/t
Mass	n/t	n/t	n/t
Mods	n/t	n/t	n/t
NIST score	n/t	n/t	n/t
P-value	n/t	n/t	n/t
Peptide	n/t	n/t	n/t
RT	n/t	n/t	n/t
Start	n/t	n/t	n/t
Stop	n/t	n/t	n/t
Theo Mass	n/t	n/t	n/t
aaScore	n/t	n/t	n/t
err(data-denovo)	n/t	n/t	n/t
gi	n/t	n/t	n/t
mz(data)	n/t	n/t	n/t
pepMass(denovo)	n/t	n/t	n/t
peptide	n/t	n/t	n/t

Ursgal Value			
.	deepnovo_style_1	flash_lfq_style_1	koja
ppm(1e6*err/(mz*z))	n/t	n/t	n/t
scanNum	n/t	n/t	n/t
score	n/t	n/t	n/t
z	n/t	n/t	n/t
# id	n/t	n/t	n/t
#Index	n/t	n/t	n/t
#SpecFile	n/t	n/t	n/t
1-lg10 EM	n/t	n/t	n/t
A_score	n/t	n/t	n/t
Alignment Score	n/t	n/t	n/t
Amanda Score	n/t	n/t	n/t
Base Sequence	n/t	Sequence	n/t
Base Sequences Mapped	n/t	FlashLFQ:Base Sequences Mapped	n/t
C-Gap	n/t	n/t	n/t
Charge	n/t	n/t	n/t
Composite Score	n/t	n/t	n/t
Context	n/t	n/t	n/t
Context Mod Variants	n/t	n/t	n/t
CoreFuc	n/t	n/t	n/t
CoreMatched	n/t	n/t	n/t
CumProb	n/t	n/t	n/t
De Novo Peptide	n/t	n/t	n/t
De Novo Score	n/t	n/t	n/t
DeNovoScore	n/t	n/t	n/t
Downstream Amino Acid	n/t	n/t	n/t
EM Probability	n/t	n/t	n/t
EValue	n/t	n/t	n/t
File Name	n/t	Raw Filename	n/t
Filename	n/t	n/t	n/t
Full Sequence	n/t	FlashLFQ:Full Sequence	n/t
Full Sequences Mapped	n/t	FlashLFQ:Full Sequences Mapped	n/t
GlyDecoy	n/t	n/t	n/t
GlyFrag	n/t	n/t	n/t
GlyID	n/t	n/t	n/t
GlyIonRatio	n/t	n/t	n/t
GlyMass	n/t	n/t	n/t
GlyScore	n/t	n/t	n/t
GlySite	n/t	n/t	n/t
GlySpec	n/t	n/t	n/t
Glycan(H,N,A,G,F)	n/t	n/t	n/t
GlycanFDR	n/t	n/t	n/t
Hit rank	n/t	n/t	n/t
Hyperscore	n/t	n/t	n/t
Intercept of expectation model (expectation in log space)	n/t	n/t	n/t
MBR Score	n/t	FlashLFQ:MBR Score	n/t
MGF_title	n/t	n/t	n/t
MS1_pearson_correlation_coefficient	n/t	n/t	n/t
MS2 Retention Time	n/t	FlashLFQ:MS2 Retention Time	n/t

Ursgal Value			
.	deepnovo_style_1	flash_lfq_style_1	koja
MSGFScore	n/t	n/t	n/t
Mass difference	n/t	n/t	n/t
MassDeviation	n/t	n/t	n/t
Matched fragment ions	n/t	n/t	n/t
Matching Tag Length	n/t	n/t	n/t
Mod	n/t	n/t	n/t
Mod Ambig Edges	n/t	n/t	n/t
Mod Ranges	n/t	n/t	n/t
Modifications	n/t	n/t	n/t
Mods	n/t	n/t	n/t
N-Gap	n/t	n/t	n/t
Neutral mass of peptide	n/t	n/t	n/t
Next score	n/t	n/t	n/t
Num Charge States Observed	n/t	FlashLFQ:Num Charge States Observed	n/t
Num Matches	n/t	n/t	n/t
Num Mod Occurrences	n/t	n/t	n/t
Number of missed cleavages	n/t	n/t	n/t
Number of tryptic termini	n/t	n/t	n/t
Obs M+H	n/t	n/t	n/t
PPM	n/t	n/t	n/t
PSMId	n/t	n/t	PSM
PSMs Mapped	n/t	FlashLFQ:PSMs Mapped	n/t
Peak Apex Mass Error (ppm)	n/t	Accuracy (ppm)	n/t
Peak Charge	n/t	FlashLFQ:Peak Charge	n/t
Peak Detection Type	n/t	FlashLFQ:Peak Detection Type	n/t
Peak MZ	n/t	FlashLFQ:Peak MZ	n/t
Peak RT Apex	n/t	FlashLFQ:Peak RT Apex	n/t
Peak RT End	n/t	FlashLFQ:Peak RT End	n/t
Peak RT Start	n/t	FlashLFQ:Peak RT Start	n/t
Peak Split Valley RT	n/t	FlashLFQ:Peak Split Valley RT	n/t
Peak intensity	n/t	FlashLFQ:Peak intensity	n/t
PepDecoy	n/t	n/t	n/t
PepIonRatio	n/t	n/t	n/t
PepScore	n/t	n/t	n/t
PepSpec	n/t	n/t	n/t
Peptide	n/t	n/t	n/t
Peptide Monoisotopic Mass	n/t	Calc mass	n/t
Peptide Sequence	n/t	n/t	n/t
PeptideFDR	n/t	n/t	n/t
PeptideMH	n/t	n/t	n/t
PlausibleStruct	n/t	n/t	n/t
PnvScr	n/t	n/t	n/t
Precursor	n/t	n/t	n/t
Precursor Charge	n/t	Charge	n/t
Precursor charge	n/t	n/t	n/t
Precursor neutral mass (Da)	n/t	n/t	n/t
PrecursorMH	n/t	n/t	n/t
PrecursorMZ	n/t	n/t	n/t

Ursgal Value			
.	deepnovo_style_1	flash_lfq_style_1	koja
Protein	n/t	n/t	n/t
Protein Accessions	n/t	n/t	n/t
Protein Group	n/t	Protein ID	n/t
Proteins	n/t	n/t	n/t
RT	n/t	n/t	n/t
Rank	n/t	n/t	n/t
RawName	n/t	n/t	n/t
Retention Time	n/t	n/t	n/t
Retention time (minutes)	n/t	n/t	n/t
RnkScr	n/t	n/t	n/t
Scan	n/t	n/t	n/t
Scan Number	n/t	n/t	n/t
ScanF	n/t	n/t	n/t
ScanID	n/t	n/t	n/t
ScanNum	n/t	n/t	n/t
Sequence	n/t	n/t	n/t
Slope of expectation model (expectation in log space)	n/t	n/t	n/t
SpecEValue	n/t	n/t	n/t
SpecFile	n/t	n/t	n/t
Spectrum Score	n/t	n/t	n/t
Spectrum number	n/t	n/t	n/t
Theo M+H	n/t	n/t	n/t
Theoretical MZ	n/t	FlashLFQ:Theoretical MZ	n/t
Title	n/t	n/t	n/t
Total possible number of matched theoretical fragment ions	n/t	n/t	n/t
TotalFDR	n/t	n/t	n/t
TotalScore	n/t	n/t	n/t
Unique Siblings	n/t	n/t	n/t
Upstream Amino Acid	n/t	n/t	n/t
Variable modifications detected	n/t	n/t	n/t
Weighted Probability	n/t	n/t	n/t
[M+H]	n/t	n/t	n/t
abs_ppm	n/t	n/t	n/t
best_locs	n/t	n/t	n/t
best_score_with_delta_mass	n/t	n/t	n/t
calc_neutral_pep_mass	n/t	n/t	n/t
charge	n/t	n/t	n/t
delta_C_n	n/t	n/t	n/t
delta_score	n/t	n/t	n/t
exp_mass	n/t	n/t	n/t
expectscore	n/t	n/t	n/t
hit_rank	n/t	n/t	n/t
hyperscore	n/t	n/t	n/t
isotope_correction	n/t	n/t	n/t
labelling	n/t	n/t	n/t
m/z	n/t	n/t	n/t
massdiff	n/t	n/t	n/t
modification_info	n/t	n/t	n/t

Ursgal Value			
.	deepnovo_style_1	flash_lfq_style_1	koja
nextscore	n/t	n/t	n/t
num_matched_ions	n/t	n/t	n/t
num_missed_cleavages	n/t	n/t	n/t
num_tol_term	n/t	n/t	n/t
other_PTM_patterns	n/t	n/t	n/t
output_aa_probs	n/t	n/t	n/t
peptide	n/t	n/t	Sequ
peptide_next_aa	n/t	n/t	n/t
peptide_prev_aa	n/t	n/t	n/t
posterior_error_prob	n/t	n/t	PEP
precursor_charge	Charge	n/t	n/t
precursor_mz	Exp m/z	n/t	n/t
precursor_neutral_mass	n/t	n/t	n/t
predicted_position_score	DeepNovo:aaScore	n/t	n/t
predicted_score	DeepNovo:score	n/t	n/t
predicted_sequence	Sequence	n/t	n/t
protein	n/t	n/t	n/t
proteinIds	n/t	n/t	Prot
protein_ID	n/t	n/t	n/t
q-value	n/t	n/t	q-va
retention_time	n/t	n/t	n/t
scan	Spectrum ID	n/t	n/t
scan_list_middle	Spectrum ID	n/t	n/t
scan_num	n/t	n/t	n/t
scannum	n/t	n/t	n/t
score	n/t	n/t	Koja
score_without_delta_mass	n/t	n/t	n/t
second_best_score_with_delta_mass	n/t	n/t	n/t
theo_mass	n/t	n/t	n/t
tot_num_ions	n/t	n/t	n/t

5.1.83 heatmap_annotation_field_name

The name of the annotation to plot in the heatmap

Default value Protein

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_annotation_field_name*

Style	Translation
heatmap_style_1	heatmap_annotation_field_name

5.1.84 heatmap_box_style

Box style for the heatmap

Default value classic

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_box_style*

Style	Translation
heatmap_style_1	heatmap_box_style

5.1.85 heatmap_color_gradient

Color gradient for the heatmap

Default value Spectral

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_color_gradient*

Style	Translation
heatmap_style_1	heatmap_color_gradient

5.1.86 heatmap_column_positions

The position of each column in the heatmap is given as a dict with keys corresponding to the position and values correspondingto the column name, e.g: {“0” : “Ratio1_2”, “1” : “Ratio2_3”}

Default value []

type dict

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_column_positions*

Style	Translation
heatmap_style_1	heatmap_column_positions

5.1.87 heatmap_error_suffix

The suffix to identify the value error holding columns

Default value _std

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_error_suffix*

Style	Translation
heatmap_style_1	heatmap_error_suffix

5.1.88 heatmap_identifier_field_name

The name of the identifier to plot in the heatmap

Default value Protein

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_identifier_field_name*

Style	Translation
heatmap_style_1	heatmap_identifier_field_name

5.1.89 heatmap_max_value

Maximum value for the color gradient

Default value 3

type int

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_max_value*

Style	Translation
heatmap_style_1	heatmap_max_value

5.1.90 heatmap_min_value

Minimum vaue for the color gradient

Default value -3

type int

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_min_value*

Style	Translation
heatmap_style_1	heatmap_min_value

5.1.91 heatmap_value_suffix

The suffix to identify the value columns, which should be plotted

Default value _mean

type str

triggers rerun True

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal key translations for *heatmap_value_suffix*

Style	Translation
heatmap_style_1	heatmap_value_suffix

5.1.92 helper_extension

Exension for helper files

Default value .u.json

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *helper_extension*

Style	Translation
ucontroller_style_1	helper_extension

5.1.93 http_output_folder

Default http download path '' : None

Default value None

type str

triggers rerun True

Available in unodes

- get_http_files_1_0_0

Ursgal key translations for *http_output_folder*

Style	Translation
get_http_style_1	http_output_folder

5.1.94 http_url

http download URL, will fail if it is not set by the user '' : None

Default value None

type str

triggers rerun True

Available in unodes

- get_http_files_1_0_0

Ursgal key translations for *http_url*

Style	Translation
get_http_style_1	http_url

5.1.95 identifier_column_names

The (combination of) specified csv column name(s) are used as identifiers. E.g. to count the number of peptides for these identifiers. The parameter "count_column_names" defines the countable elements.

Default value ['Protein ID']

type list

triggers rerun True

Available in unodes

- csv2counted_results_1_0_0

Ursgal key translations for *identifier_column_names*

Style	Translation
csv2counted_results_style_1	identifier_column_names

5.1.96 infer_proteins

Use the picked-protein algorithm to infer protein PEP and FDR in Percolator

Default value False

type bool

triggers rerun True

Available in unodes

- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0

Ursgal key translations for *infer_proteins*

Style	Translation
percolator_style_1	infer_proteins

5.1.97 instrument

Type of mass spectrometer (used to determine the scoring model)

Default value q_exactive

type select

triggers rerun True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979

- novor_1_1beta
- novor_1_05

Ursgal key translations for *instrument*

Style	Translation
kojak_style_1	instrument
moda_style_1	Instrument
msgfplus_style_1	-inst
novor_style_1	massAnalyzer

Ursgal value translations

Ursgal Value	Translated Value			
.	kojak_style_1	moda_style_1	msgfplus_style_1	novor_style_1
FTICR	1	n/t	n/t	n/t
high_res_ltq	0	ESI-TRAP	1	Trap
low_res_ltq	0	ESI-TRAP	0	Trap
q_exactive	0	ESI-TRAP	3	FT
tof	n/t	ESI-QTOF	2	TOF

5.1.98 integrate_peak_areas

integrate peak areas

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *integrate_peak_areas*

Style	Translation
flash_lfq_style_1	–int

5.1.99 intensity_cutoff

Low intensity cutoff as a fraction of max peak

Default value 0.0

type float

triggers rerun True

Available in unodes

- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- ptmshepherd_0_3_5

Ursgal key translations for *intensity_cutoff*

Style	Translation
msfragger_style_1	minimum_ratio
msfragger_style_2	minimum_ratio
msfragger_style_3	minimum_ratio
omssa_style_1	-cl
ptmshepherd_style_1	spectra_condRatio

5.1.100 intensity_cutoff_diagnostic_ions

Minimum relative intensity (relative to base peak height) for the sum of all diagnostic fragment ion intensities

Default value 0.0

type float

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *intensity_cutoff_diagnostic_ions*

Style	Translation
msfragger_style_3	diagnostic_intensity_filter

5.1.101 intensity_transformation_factor

Tranform intensity by this factor for quantification

Default value 100000.0

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *intensity_transformation_factor*

Style	Translation
pyqms_style_1	INTENSITY_TRANSFORMATION_FACTOR
sugarpy_plot_style_1	INTENSITY_TRANSFORMATION_FACTOR
sugarpy_run_style_1	INTENSITY_TRANSFORMATION_FACTOR

5.1.102 internal_precision

Float to int conversion precision

Default value 1000.0

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- glycopeptide_fragmentor_1_0_0

Ursgal key translations for *internal_precision*

Style	Translation
glycopeptide_fragmentor_style_1	internal_precision
pyqms_style_1	INTERNAL_PRECISION
sugarpy_plot_style_1	INTERNAL_PRECISION
sugarpy_run_style_1	INTERNAL_PRECISION

5.1.103 ion_mode

The ion mode that has been used for acquiring mass spectra (positive or negative)

Default value positive

type select

triggers rerun True

Available in unodes

- mzml2mgf_2_0_0

Ursgal key translations for *ion_mode*

Style	Translation
mzml2mgf_style_1	ion_mode

Ursgal value translations

Ursgal Value	Translated Value
.	mzml2mgf_style_1
negative	•
positive	•

5.1.104 is_open_search

specifying whether this is an open search or not

Default value True

type bool

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *is_open_search*

Style	Translation
ptminer_style_1	open_search

Ursgal value translations

Ursgal Value	Translated Value
.	ptminer_style_1
0	0
1	1

5.1.105 isotopic_distribution_tolerance

isotopic distribution tolerance in ppm

Default value 5

type float

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *isotopic_distribution_tolerance*

Style	Translation
flash_lfq_style_1	–iso

5.1.106 json_extension

Exension for .json files

Default value .u.json

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *json_extension*

Style	Translation
ucontroller_style_1	json_extension

5.1.107 keep_asp_pro_broken_peps

X!tandem searches for peptides broken between Asp (D) and Pro (P) for every enzyme. Therefore, it reports peptides that are not enzymatically cleaved. Specify, if those should be kept during unify_csv or removed.

Default value True

type bool

triggers rerun True

Available in unodes

- unify_csv_1_0_0

Ursgal key translations for *keep_asp_pro_broken_peps*

Style	Translation
unify_csv_style_1	keep_asp_pro_broken_peps

5.1.108 keep_column_names

List of column headers which are are not used as identifiers but kept in the output, e.g. when counting ["Sequence", "Modifications"] the column ["Protein ID"] could be specified here. Multiple entries for one identifier (e.g. when identifier_column_names = ["Potein ID"] and keep_column_names = ["Sequence"]) are seperated by "<#>".

Default value ['Protein ID']

type list

triggers rerun True

Available in unodes

- csv2counted_results_1_0_0

Ursgal key translations for *keep_column_names*

Style	Translation
csv2counted_results_style_1	keep_column_names

5.1.109 kernel

The kernel function of the support vector machine used for PSM post-processing ('rbf', 'linear', 'poly' or 'sigmoid')

Default value rbf

type select

triggers rerun True

Available in unodes

- svm_1_0_0

Ursgal key translations for *kernel*

Style	Translation
svm_style_1	kernel

5.1.110 kojak_diff_mods_on_xl

To search differential modifications on cross-linked peptides: diff_mods_on_xl = 1

Default value 0

type int

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_diff_mods_on_xl*

Style	Translation
kojak_style_1	kojak_diff_mods_on_xl

5.1.111 kojak_enrichment

Values between 0 and 1 to describe 18O APE For example, 0.25 equals 25 APE

Default value 0

type float

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_enrichment*

Style	Translation
kojak_style_1	kojak_enrichment

5.1.112 kojak_export_pepxml

Activate (True) or deactivate (False) output as pepXML

Default value False

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_export_pepxml*

Style	Translation
kojak_style_1	kojak_export_pepXML

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.113 kojak_export_percolator

Activate (True) or deactivate (False) output for percolator

Default value True

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_export_percolator*

Style	Translation
kojak_style_1	kojak_export_percolator

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.114 kojak_fragment_bin_offset

fragment_bin_offset and fragment_bin_size influence algorithm precision and memory usage. They should be set appropriately for the data analyzed. For ion trap ms/ms: 1.0005 size, 0.4 offset For high res ms/ms: 0.03 size, 0.0 offset

Default value 0.0

type float

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_fragment_bin_offset*

Style	Translation
kojak_style_1	kojak_fragment_bin_offset

5.1.115 kojak_fragment_bin_size

fragment_bin_offset and fragment_bin_size influence algorithm precision and memory usage. They should be set appropriately for the data analyzed. For ion trap ms/ms: 1.0005 size, 0.4 offset For high res ms/ms: 0.03 size, 0.0 offset

Default value 0.03

type float

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_fragment_bin_size*

Style	Translation
kojak_style_1	kojak_fragment_bin_size

5.1.116 kojak_mono_links_on_xl

To search for mono-linked cross-linker on cross-linked peptides: mono_links_on_xl = 1

Default value 0

type int

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_mono_links_on_xl*

Style	Translation
kojak_style_1	kojak_mono_links_on_xl

5.1.117 kojak_percolator_version

Defines the output format of Kojak for Percolator

Default value 2.08

type str

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_percolator_version*

Style	Translation
kojak_style_1	kojak_percolator_version

5.1.118 kojak_prefer_precursor_pred

prefer precursor mono mass predicted by instrument software. Available values:

ignore_previous: previous predictions are ignored

only_previous: only previous predictions are used

supplement: predictions are supplemented with additional analysis

Default value supplement

type select

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_prefer_precursor_pred*

Style	Translation
kojak_style_1	kojak_prefer_precursor_pred

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
ignore_previous	0
only_previous	1
supplement	2

5.1.119 kojak_spectrum_processing

True, if spectrum should be processed by kojak

Default value False

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_spectrum_processing*

Style	Translation
kojak_style_1	kojak_spectrum_processing

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.120 kojak_top_count

number of top scoring single peptides to combine in relaxed analysis

Default value 300

type int

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_top_count*

Style	Translation
kojak_style_1	kojak_top_count

5.1.121 kojak_turbo_button

Generally speeds up analysis. Special cases cause reverse effect, thus this is allowed to be disabled. True if it should be used.

Default value False

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *kojak_turbo_button*

Style	Translation
kojak_style_1	kojak_turbo_button

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.122 label

15N if the corresponding amino acid labeling was applied

Default value 14N

type select

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242

- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5

- pipi_1_4_6
- pyqms_1_0_0

Ursgal key translations for *label*

Style	Translation
moda_style_1	label
msamanda_style_1	label
msfragger_style_1	label
msfragger_style_2	label
msfragger_style_3	label
msgfplus_style_1	label
myrimatch_style_1	label
omssa_style_1	('-tem', '-tom')
pipi_style_1	15N
pyqms_style_1	label
xtandem_style_1	protein, modified residue mass file

Ursgal value translations

Ursgal Value	Translated Value
.	pipi_style_1
14N	0
15N	1

5.1.123 label_percentile

Enrichment level of the label

Default value [0.0]

type list

triggers rerun True

Available in unodes

- pyqms_1_0_0

Ursgal key translations for *label_percentile*

Style	Translation
pyqms_style_1	label_percentile

5.1.124 label_percentile_format_string

Defines the standard format string when formatting labeling percentile float

Default value {0:.3f}

type str

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *label_percentile_format_string*

Style	Translation
pyqms_style_1	PERCENTILE_FORMAT_STRING
sugarpy_plot_style_1	PERCENTILE_FORMAT_STRING
sugarpy_run_style_1	PERCENTILE_FORMAT_STRING

5.1.125 localize_delta_mass

Generate and use mass difference fragment index in addition to the regular fragment index for search. This allows shifted fragment ions - fragment ions with mass increased by the calculated mass difference, to be included in scoring.

Default value True

type bool

triggers rerun True

Available in unodes

- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *localize_delta_mass*

Style	Translation
msfragger_style_1	localize_delta_mass
msfragger_style_2	localize_delta_mass
msfragger_style_3	localize_delta_mass

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
0	0	0	0
1	1	1	1

5.1.126 lower_mz_limit

lowest considered mz for quantification

Default value 150

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *lower_mz_limit*

Style	Translation
pyqms_style_1	LOWER_MZ_LIMIT
sugarpy_plot_style_1	LOWER_MZ_LIMIT
sugarpy_run_style_1	LOWER_MZ_LIMIT

5.1.127 m_score_cutoff

minimum required pyQms m_score for a quant event to be evaluated

Default value 0.7

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *m_score_cutoff*

Style	Translation
pyqms_style_1	M_SCORE_THRESHOLD
sugarpy_plot_style_1	M_SCORE_THRESHOLD
sugarpy_run_style_1	M_SCORE_THRESHOLD

5.1.128 machine_offset_in_ppm

Machine offset, m/z values will be corected/shifted by the given value.

Default value 0.0

type float

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *machine_offset_in_ppm*

Style	Translation
mzml2mgf_style_1	machine_offset_in_ppm
pyqms_style_1	MACHINE_OFFSET_IN_PPM
sugarpy_plot_style_1	MACHINE_OFFSET_IN_PPM
sugarpy_run_style_1	MACHINE_OFFSET_IN_PPM

5.1.129 markov_chain_burn_in_iterations

number of markov-chain monte carlo burn in iterations for the Bayesian protein fold-change analysis

Default value 1000

type int

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *markov_chain_burn_in_iterations*

Style	Translation
flash_lfq_style_1	–bur

5.1.130 markov_chain_iterations

number of markov-chain monte carlo iterations for the Bayesian protein fold-change analysis

Default value 3000

type int

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *markov_chain_iterations*

Style	Translation
flash_lfq_style_1	–mcm

5.1.131 mass_diff_bin_size

Number of bins per dalton to be used for mass shift binning. That means 5000 bins correspond to a bin size of 0.0002 Da

Default value 5000

type int

triggers rerun True

Available in unodes

- ptmshepherd_0_3_5

Ursgal key translations for *mass_diff_bin_size*

Style	Translation
ptmshepherd_style_1	histo_bindivs

5.1.132 mass_offset_list

List of mass offsets at which modification peaks will be checked for (e.g. [0, 79.9663])

Default value []

type list

triggers rerun True

Available in unodes

- ptmshepherd_0_3_5

Ursgal key translations for *mass_offset_list*

Style	Translation
ptmshepherd_style_1	mass_offsets

5.1.133 match_between_runs

Quantify PSMs identified in other runs

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *match_between_runs*

Style	Translation
flash_lfq_style_1	–mbr

5.1.134 match_between_runs_RT_window

Max RT differenence in minutes of peptides to be considered for MBR

Default value 1

type float

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *match_between_runs_RT_window*

Style	Translation
flash_lfq_style_1	–mrt

5.1.135 max_accounted_observed_peaks

Maximum number of peaks from a spectrum used.

Default value 150

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- myrimatch_2_1_138
- myrimatch_2_2_140
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- deepnovo_pointnovo
- ptmshepherd_0_3_5

Ursgal key translations for *max_accounted_observed_peaks*

Style	Translation
deepnovo_style_1	MAX_NUM_PEAK
kojak_style_1	max_accounted_observed_peaks
msfragger_style_1	use_topN_peaks
msfragger_style_2	use_topN_peaks
msfragger_style_3	use_topN_peaks
myrimatch_style_1	MaxPeakCount
ptmshepherd_style_1	spectra_condPeaks
xtandem_style_1	spectrum, total peaks

5.1.136 max_glycan_length

Maximum number of monosaccharides per glycan

Default value 15

type int

triggers rerun False

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *max_glycan_length*

Style	Translation
sugarpy_plot_style_1	max_tree_length
sugarpy_run_style_1	max_tree_length

5.1.137 max_missed_cleavages

Maximum number of missed cleavages per peptide

Default value 2

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- unify_csv_1_0_0
- upeptide_mapper_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03

- pglyco_db_2_2_0
- pglyco_db_2_2_2
- deepnovo_0_0_1

Ursgal key translations for *max_missed_cleavages*

Style	Translation
deepnovo_style_1	num_missed_cleavage
kojak_style_1	max_missed_cleavages
moda_style_1	MissedCleavage
msamanda_style_1	missed_cleavages
msfragger_style_1	allowed_missed_cleavage
msfragger_style_2	allowed_missed_cleavage
msfragger_style_3	allowed_missed_cleavage
msgfplus_style_1	-maxMissedCleavages
myrimatch_style_1	MaxMissedCleavages
omssa_style_1	-v
pglyco_db_style_1	max_miss_cleave
pipi_style_1	missed_cleavage
unify_csv_style_1	max_missed_cleavages
upeptide_mapper_style_1	max_missed_cleavages
xtandem_style_1	scoring, maximum missed cleavage sites

5.1.138 max_mod_alternatives

Maximal number of variable modification alternatives, given as C in 2^C

Default value 6

type int

triggers rerun True

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *max_mod_alternatives*

Style	Translation
xtandem_style_1	protein, ptm complexity

5.1.139 max_mod_size

Maximum modification size to consider (in Da)

Default value 200

type int

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- pipi_1_4_5
- pipi_1_4_6
- ptminer_1_0

Ursgal key translations for *max_mod_size*

Style	Translation
moda_style_1	MaxModSize
pipi_style_1	max_ptm_mass
ptminer_style_1	precursor_matching_tolerance

5.1.140 max_molecules_per_match_bin

Max number of molecules in one matching bin.

Default value 5000

type int

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *max_molecules_per_match_bin*

Style	Translation
pyqms_style_1	MAX_MOLECULES_PER_MATCH_BIN
sugarpy_plot_style_1	MAX_MOLECULES_PER_MATCH_BIN
sugarpy_run_style_1	MAX_MOLECULES_PER_MATCH_BIN

5.1.141 max_num_mod_sites

Maximum number of potential modification sites for a specific modification per peptide. Peptides with a higher number are discarded, due to a too high complexity.

Default value 6

type int

triggers rerun True

Available in unodes

- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665

Ursgal key translations for *max_num_mod_sites*

Style	Translation
msamanda_style_1	MaxNumberModSites

5.1.142 max_num_mods

Maximal number of modifications per peptide

Default value 3

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03

- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- msamanda_2_0_0_9695
- msamanda_2_0_0_9706
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- pglyco_db_2_2_0
- pglyco_db_2_2_2

Ursgal key translations for *max_num_mods*

Style	Translation
kojak_style_1	max_num_mods
msamanda_style_1	MaxNoDynModifs
msgfplus_style_1	NumMods
myrimatch_style_1	MaxDynamicMods
pglyco_db_style_1	max_var_modify_num

5.1.143 max_num_neutral_loss

Maximum number of same neutral losses per peptide regarding water and ammonia losses.

Default value 1

type int

triggers rerun True

Available in unodes

- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665

Ursgal key translations for *max_num_neutral_loss*

Style	Translation
msamanda_style_1	MaxNumberNeutralLoss

5.1.144 max_num_neutral_loss_mod

Maximum number of same neutral losses per peptide regarding modification specific losses.

Default value 2

type int

triggers rerun True

Available in unodes

- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665

Ursgal key translations for *max_num_neutral_loss_mod*

Style	Translation
msamanda_style_1	MaxNumberNeutralLossModificati

5.1.145 max_num_of_ions_per_series_to_search

Max number of ions in each series being searched 0 : all

Default value 0

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *max_num_of_ions_per_series_to_search*

Style	Translation
omssa_style_1	-sp

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
all	0

5.1.146 max_num_per_mod

Maximum number of residues that can be occupied by each variable modification (maximum of 5)

Default value 3

type int

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665

Ursgal key translations for *max_num_per_mod*

Style	Translation
msamanda_style_1	MaxNoModifs
msfragger_style_1	max_variable_mods_per_mod
msfragger_style_2	max_variable_mods_per_mod
msfragger_style_3	max_variable_mods_per_peptide

5.1.147 max_num_per_mod_name_specific

Maximal number of modification sites per peptide for a specific modification, given as a dictionary:
{unimod_name : number}

Default value []

type dict

triggers rerun True

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *max_num_per_mod_name_specific*

Style	Translation
xtandem_style_1	residue, potential modification mass

5.1.148 max_num_psms_per_spec

Maximum number of PSMs retained per spectrum

Default value 30

type int

triggers rerun True

Available in unodes

- omssa_2_1_9
- sanitize_csv_1_0_0

Ursgal key translations for *max_num_psms_per_spec*

Style	Translation
omssa_style_1	-hl
sanitize_csv_style_1	max_output_psms

5.1.149 max_num_substring_mod_pep

Maximum number of times a de novo-produced substring can occur in the protein sequence database for TagGraph to consider it as a modified peptide match

Default value 200

type int

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *max_num_substring_mod_pep*

Style	Translation
tag_graph_style_1	modmaxcounts

5.1.150 max_num_substring_pep

Maximum number of times a de novo-produced substring can occur in the protein sequence database for TagGraph to consider it as a modified peptide match

Default value 400

type int

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *max_num_substring_pep*

Style	Translation
tag_graph_style_1	maxcounts

5.1.151 max_output_e_value

Highest e-value for reported peptides

Default value 1.0

type float

triggers rerun True

Available in unodes

- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *max_output_e_value*

Style	Translation
omssa_style_1	-he
xtandem_style_1	output, maximum valid expectation value

5.1.152 max_pep_length

Maximal length of a peptide

Default value 40

type int

triggers rerun True

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pglyco_db_2_2_0
- pglyco_db_2_2_2

Ursgal key translations for *max_pep_length*

Style	Translation
msfragger_style_1	digest_max_length
msfragger_style_2	digest_max_length
msfragger_style_3	digest_max_length
msgfplus_style_1	-maxLength
myrimatch_style_1	MaxPeptideLength
omssa_style_1	-nox
pglyco_db_style_1	max_peptide_len
pipi_style_1	max_peptide_length

5.1.153 max_pep_var

Maximal peptide variants, new default defined by msfragger

Default value 1000

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *max_pep_var*

Style	Translation
msfragger_style_1	max_variable_mods_combinations
msfragger_style_2	max_variable_mods_combinations
msfragger_style_3	max_variable_mods_combinations
msgfplus_style_1	-maxLength
myrimatch_style_1	MaxPeptideVariants
omssa_style_1	-nox

5.1.154 max_protein_name

Max protein name for output. For kojak, this defines the number of character (0=off), for TagGraph the number of protein names

Default value 5

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- tag_graph_1_8_0

Ursgal key translations for *max_protein_name*

Style	Translation
kojak_style_1	kojak_truncate_prot_names
tag_graph_style_1	DisplayProtNum

5.1.155 max_trees_per_spec

Max number of glycoforms reported per spectrum for each peptide

Default value 5

type int

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *max_trees_per_spec*

Style	Translation
sugarpy_plot_style_1	max_trees_per_spec
sugarpy_run_style_1	max_trees_per_spec

5.1.156 mgf_input_file

Path to input .mgf file '' : None

Default value None

type str

triggers rerun True

Available in unodes

- deepnovo_0_0_1
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- pnovo_3_1_3
- xtandem_cyclone_2010

- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *mgf_input_file*

Style	Translation
deepnovo_style_1	('denovo_input_file', 'hybrid_input_file', 'db_input_file')
moda_style_1	Spectra
msamanda_style_1	mgf_input_file
msgfplus_style_1	-s
novor_style_1	-f
omssa_style_1	-fm
pepnovo_style_1	-file
pglyco_db_style_1	file1
pnovo_style_1	spec_path1
xtandem_style_1	spectrum, path

5.1.157 mgf_input_files_list

List of paths to input .mgf files '' : None

Default value None

type list

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *mgf_input_files_list*

Style	Translation
ptminer_style_1	mgf_input_files_list

5.1.158 min_element_abundance

Set minmal abundance for elements used when building isotopologue library

Default value 0.001

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *min_element_abundance*

Style	Translation
pyqms_style_1	ELEMENT_MIN_ABUNDANCE
sugarpy_plot_style_1	ELEMENT_MIN_ABUNDANCE
sugarpy_run_style_1	ELEMENT_MIN_ABUNDANCE

5.1.159 min_glycan_length

Minimum number of monosaccharides per glycan

Default value 3

type int

triggers rerun False

Available in unodes

- sugarpy_run_1_0_0

Ursgal key translations for *min_glycan_length*

Style	Translation
sugarpy_run_style_1	min_tree_length

5.1.160 min_mod_size

Minimum modification size to consider (in Da)

Default value -200

type int

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- pipi_1_4_5

- pipi_1_4_6
- ptminer_1_0

Ursgal key translations for *min_mod_size*

Style	Translation
moda_style_1	MinModSize
pipi_style_1	min_ptm_mass
ptminer_style_1	precursor_matching_tolerance

5.1.161 min_num_mods

Minimal number of modifications per peptide

Default value 5

type int

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *min_num_mods*

Style	Translation
ptminer_style_1	min_mod_number

5.1.162 min_number_of_matched_isotopologues

Min number of matched isotopologues to consider for quantification

Default value 2

type int

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- flash_lfq_1_1_1

Ursgal key translations for *min_number_of_matched_isotopologues*

Style	Translation
flash_lfq_style_1	–nis
pyqms_style_1	MINIMUM_NUMBER_OF_MATCHED_ISOTOPOLOGUES
sugarpy_plot_style_1	MINIMUM_NUMBER_OF_MATCHED_ISOTOPOLOGUES
sugarpy_run_style_1	MINIMUM_NUMBER_OF_MATCHED_ISOTOPOLOGUES

5.1.163 min_number_of_spectra

Min number of spectra in which a molecule needs to be matched in order to consider it for further processing

Default value 2

type int

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *min_number_of_spectra*

Style	Translation
sugarpy_plot_style_1	min_spec_number
sugarpy_run_style_1	min_spec_number

5.1.164 min_output_score

Lowest score for reported peptides. If set to '-1e-10', default values fo each engine will be used. -1e-10 = 'default'

Default value -1e-10

type float

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- pepnovo_3_1

Ursgal key translations for *min_output_score*

Style	Translation
myrimatch_style_1	MinResultScore
pepnovo_style_1	-min_filter_prob

Ursgal value translations

Ursgal Value	Translated Value	
.	myrimatch_style_1	pepnovo_style_1
-1e-10	1e-07	0.9

5.1.165 min_oxonium_ions

Min number of oxonium ions that need to be matched in an MS/MS spectrum, to be accepted as containing oxonium ions (i.e. considered as glycopeptide)

Default value 3

type int

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0
- glycopeptide_fragmentor_1_0_0

Ursgal key translations for *min_oxonium_ions*

Style	Translation
glycopeptide_fragmentor_style_1	min_oxonium_ions
sugarpy_plot_style_1	min_oxonium_ions

5.1.166 min_pep_length

Minimal length of a peptide

Default value 6

type int

triggers rerun True

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- pipi_1_4_5
- pipi_1_4_6
- pglyco_db_2_2_0
- pglyco_db_2_2_2

Ursgal key translations for *min_pep_length*

Style	Translation
msamanda_style_1	MinimumPepLength
msfragger_style_1	digest_min_length
msfragger_style_2	digest_min_length
msfragger_style_3	digest_min_length
msgfplus_style_1	-minLength
myrimatch_style_1	MinPeptideLength
omssa_style_1	-no
pglyco_db_style_1	min_peptide_len
pipi_style_1	min_peptide_length

5.1.167 min_precursor_matches

Minimum number of precursors that match a spectrum.

Default value 1

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *min_precursor_matches*

Style	Translation
omssa_style_1	-pc

5.1.168 min_required_matched_peaks

Mimimum number of matched ions required for a peptide to be scored, MSFragger default: 4

Default value 4

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer

- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *min_required_matched_peaks*

Style	Translation
msfragger_style_1	min_matched_fragments
msfragger_style_2	min_matched_fragments
msfragger_style_3	min_matched_fragments
myrimatch_style_1	MinMatchedFragments
omssa_style_1	-hm
xtandem_style_1	scoring, minimum ion count

5.1.169 min_required_observed_peaks

Mimimum number of peaks in the spectrum to be considered. MSFragger default: 15

Default value 5

type int

triggers rerun True

Available in unodes

- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222

- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *min_required_observed_peaks*

Style	Translation
msfragger_style_1	minimum_peaks
msfragger_style_2	minimum_peaks
msfragger_style_3	minimum_peaks
omssa_style_1	-hs
xtandem_style_1	spectrum, minimum peaks

5.1.170 min_subtree_coverage

Min subtree coverage to be considered for output

Default value 0.6

type float

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *min_subtree_coverage*

Style	Translation
sugarpy_plot_style_1	min_sub_cov
sugarpy_run_style_1	min_sub_cov

5.1.171 min_sugarpy_score

Min SugarPy score to be considered for output

Default value 1.0

type float

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *min_sugarpy_score*

Style	Translation
sugarpy_plot_style_1	min_sugarpy_score
sugarpy_run_style_1	min_sugarpy_score

5.1.172 min_y_ions

Min number of Y-ions that need to be matched in an MS/MS spectrum, to be accepted as containing Y-ions (i.e. considered as glycopeptide)

Default value 1

type int

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0
- glycopeptide_fragmentor_1_0_0

Ursgal key translations for *min_y_ions*

Style	Translation
glycopeptide_fragmentor_style_1	min_Y_ions
sugarpy_plot_style_1	min_Y_ions

5.1.173 moda_blind_mode

Allowed number of modifications per peptide in ModA BlindMode. Available values: no_modification

one_modification no_limit

Default value no_limit

type select

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62

Ursgal key translations for *moda_blind_mode*

Style	Translation
moda_style_1	BlindMode

Ursgal value translations

Ursgal Value	Translated Value
.	moda_style_1
no_limit	2
no_modification	0
one_modification	1

5.1.174 moda_high_res

If True, fragment tolerance is set as the same as precursor tolerance, when the peptide mass is significantly small, such that fragment tolerance is larger than precursor tolerance

Default value True

type bool

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62

Ursgal key translations for *moda_high_res*

Style	Translation
moda_style_1	HighResolution

Ursgal value translations

Ursgal Value	Translated Value
.	moda_style_1
0	OFF
1	ON

5.1.175 moda_protocol_id

MODa specific protocol to enable scoring parameters for labeled samples.

Default value None

type select

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62

Ursgal key translations for *moda_protocol_id*

Style	Translation
moda_style_1	Protocol

Ursgal value translations

Ursgal Value	Translated Value
.	moda_style_1
None	NONE

5.1.176 modification_mass_tolerance

Maximum absolute deviation (Da) between experimental and database modification mass

Default value 0.1

type float

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *modification_mass_tolerance*

Style	Translation
tag_graph_style_1	modtolerance

5.1.177 modifications

Modifications are given as a list of strings, each representing the modification of one amino acid. The string consists of four informations seperated by comma:

> 'amino acid, type, position, unimod name or id'
>
> amino acid : specify the modified amino acid as a single letter, use '*' if the amino acid is variable type : specify if it is a fixed (fix) or potential (opt) modification position : specify the position within the protein/peptide (Prot-N-term, Prot-C-term), use 'any' if the positon is variable unimod name or id: specify the unimod PSI-MS Name or unimod Accession # (see unimod.org)

Examples:

> ['M,opt,any,Oxidation'] - potential oxidation of Met at any position within a peptide ['*,opt,Prot-N-term,Acetyl'] - potential acetylation of any amino acid at the N-terminus of a protein ['S,opt,any,Phospho'] - potential phosphorylation of Serine at any position within a peptide ['C,fix,any,Carbamidomethyl', 'N,opt,any,Deamidated', 'Q,opt,any,Deamidated'] - fixed carbamidomethylation of Cys and potential deamidation of Asn and/or Gln at any position within a peptide

Additionally, userdefined modifications can be given and are written to a userdefined_unimod.xml in ursgal/kb/ext. Userdefined modifications need to have a unique name instead of the unimod name the chemical composition needs to be given as a Hill notation on the fifth position in the string

Example:

['S,opt,any,New_mod,C2H5N1O3']

> **Default value** ['*,opt,Prot-N-term,Acetyl', 'M,opt,any,Oxidation']
>
> **type** list
>
> **triggers rerun** True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723

- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- unify_csv_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pyqms_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- deepnovo_0_0_1

- pnovo_3_1_3
- tag_graph_1_8_0
- flash_lfq_1_1_1
- ptmshepherd_0_3_5

Ursgal key translations for *modifications*

Style	Translation
deep-novo_style_1	modifications
flash_lfq_style_1	modifications
ko-jak_style_1	modifications
moda_style_1	AMD
msamanda_style_1	modifications
ms-frag-ger_style_1	modifications
ms-frag-ger_style_2	modifications
ms-frag-ger_style_3	modifications
msgf-plus_style_1	-mod
myri-match_style_1	(‘DynamicMods’, ‘StaticMods’)
novor_style_1	(‘variableModifications’, ‘fixedModifications’)
omssa_style_1	(‘-mv’, ‘mf’)
pep-novo_style_1	-PTMs
pg-lyco_db_style_1	modifications
pipi_style_1	modifications
pnovo_style_1	modifications
pt-miner_style_1	modifications
ptmshep-herd_style_1	varmod_masses
pyqms_style_1	modifications
tag_graph_style_1	modifications
unify_csv_style_1	modifications
xtan-dem_style_1	(‘residue, modification mass’, ‘residue, potential modification mass’, ‘protein, N-terminal residue modification mass’, ‘protein, C-terminal residue modification mass’, ‘protein, C-terminal residue modification mass’, ‘protein, quick acetyl’, ‘protein, quick pyrolidone’)

5.1.178 modifications_offsets

Specify molecules (or masses) that will be used as mass offsets in the search. Specify as a dictionary with the keys "chemical_formulas", "unimods", "glycans", "masses", and lists with the corresponding molecules as values.

Default value ['chemical_formulas', 'glycans', 'masses', 'unimods']

type dict

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *modifications_offsets*

Style	Translation
msfragger_style_3	mass_offsets

5.1.179 modifications_y_ion_offsets

Specify molecules (or masses) that will be used as mass offsets for Y-ions in the search. Specify as a dictionary with the keys "chemical_formulas", "unimods", "glycans", "masses", and lists with the corresponding molecules as values.

Default value ['chemical_formulas', 'glycans', 'masses', 'unimods']

type dict

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *modifications_y_ion_offsets*

Style	Translation
msfragger_style_3	Y_type_masses

5.1.180 molecules_to_quantify

Molecules to quantify. Can be either a list of strings or a csv file

Default value None

type list

triggers rerun True

Available in unodes

- pyqms_1_0_0

Ursgal key translations for *molecules_to_quantify*

Style	Translation
pyqms_style_1	molecules

5.1.181 mono_link_definition

Cross-link and mono-link masses allowed. May have more than one of each parameter. Format for mono_link is:

[amino acids] [mass mod]

One or more amino acids (uppercase only!!) can be specified for each linkage moiety. Use lowercase 'n' or 'c' to indicate protein N-terminus or C-terminus

Default value nK 156.0786

type str

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *mono_link_definition*

Style	Translation
kojak_style_1	mono_link_definition

5.1.182 monosaccharide_compositions

Dictionary defining the chemical formula (hill notation) for each monosaccharide that is used.

Default value ['Hex', 'HexNAc', 'NeuAc', 'Pent', 'dHex']

type dict

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *monosaccharide_compositions*

Style	Translation
sugarpy_plot_style_1	monosaccharides
sugarpy_run_style_1	monosaccharides

5.1.183 ms1_is_centroided

MS1 are centroided data: True or False

Default value False

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *ms1_is_centroided*

Style	Translation
kojak_style_1	kojak_MS1_centroid

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.184 ms1_resolution

MS1 resolution

Default value 30000

type int

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *ms1_resolution*

Style	Translation
kojak_style_1	kojak_MS1_resolution

5.1.185 ms2_is_centroided

MS2 are centroided data: True or False

Default value True

type bool

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *ms2_is_centroided*

Style	Translation
kojak_style_1	kojak_MS2_centroid

Ursgal value translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

5.1.186 ms2_resolution

MS2 resolution

Default value 25000

type int

triggers rerun True

Available in unodes

- kojak_1_5_3

Ursgal key translations for *ms2_resolution*

Style	Translation
kojak_style_1	kojak_MS2_resolution

5.1.187 ms_level

MS level on which that is taken into account, e.g. for spectrum extraction, matching of evidences, etc.

Default value 2

type int

triggers rerun True

Available in unodes

- pyqms_1_0_0
- mzml2mgf_1_0_0
- mzml2mgf_2_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pipi_1_4_5
- pipi_1_4_6

Ursgal key translations for *ms_level*

Style	Translation
mzml2mgf_style_1	ms_level
pipi_style_1	ms_level
pyqms_style_1	ms_level
sugarpy_plot_style_1	ms_level
sugarpy_run_style_1	ms_level

5.1.188 msfragger_add_topN_complementary

Inserts complementary ions corresponding to the top N most intense fragments in each experimental spectrum. Useful for recovery of modified peptides near C-terminal in open search. Should be set to 0 (disabled) otherwise.

Default value False

type bool

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_add_topN_complementary*

Style	Translation
msfragger_style_1	add_topN_complementary
msfragger_style_2	add_topN_complementary
msfragger_style_3	add_topN_complementary

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
0	0	0	0
1	1	1	1

5.1.189 msfragger_labile_mode

“off” corresponds to a standard MSFragger search, “labile” allows to specify delta masses that are searched for, and “nglycan” additionally checks for N-glycosylation motifs

Default value off

type select

triggers rerun True

Available in unodes

- msfragger_3_0

Ursgal key translations for *msfragger_labile_mode*

Style	Translation
msfragger_style_3	labile_search_mode

5.1.190 msfragger_min_fragments_modelling

Minimum number of matched peaks in PSM for inclusion in statistical modeling

Default value 2

type int

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_min_fragments_modelling*

Style	Translation
msfragger_style_1	min_fragments_modelling
msfragger_style_2	min_fragments_modelling
msfragger_style_3	min_fragments_modelling

5.1.191 msfragger_output_max_expect

Suppresses reporting of PSM if top hit has expectation greater than this threshold

Default value 50

type int

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_output_max_expect*

Style	Translation
msfragger_style_1	output_max_expect
msfragger_style_2	output_max_expect
msfragger_style_3	output_max_expect

5.1.192 msfragger_track_zero_topN

Track top N unmodified peptide results separately from main results internally for boosting features. Should be set to a number greater than output_report_topN if zero bin boosting is desired.

Default value 0

type int

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_track_zero_topN*

Style	Translation
msfragger_style_1	track_zero_topN
msfragger_style_2	track_zero_topN
msfragger_style_3	track_zero_topN

5.1.193 msfragger_zero_bin_accept_expect

Ranks a zero-bin hit above all non-zero-bin hit if it has expectation less than this value.

Default value 0.0

type float

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_zero_bin_accept_expect*

Style	Translation
msfragger_style_1	zero_bin_accept_expect
msfragger_style_2	zero_bin_accept_expect
msfragger_style_3	zero_bin_accept_expect

5.1.194 msfragger_zero_bin_mult_expect

Multiplies expect value of PSMs in the zero-bin during results ordering (set to less than 1 for boosting)

Default value 1.0

type float

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *msfragger_zero_bin_mult_expect*

Style	Translation
msfragger_style_1	zero_bin_mult_expect
msfragger_style_2	zero_bin_mult_expect
msfragger_style_3	zero_bin_mult_expect

5.1.195 msgfplus_mzid_converter_version

Determines which msgfplus mzid conversion node should be used e.g. "msgfplus2csv_v2017_07_04"

Default value None

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *msgfplus_mzid_converter_version*

Style	Translation
ucontroller_style_1	msgfplus_mzid_converter_version

Ursgal value translations

Ursgal Value	Translated Value
.	ucontroller_style_1
msgfplus_v2016_09_16	msgfplus2csv_py_v1_0_0
msgfplus_v2017_01_27	msgfplus2csv_py_v1_0_0
msgfplus_v2018_01_30	msgfplus2csv_py_v1_0_0
msgfplus_v2018_06_28	msgfplus2csv_py_v1_0_0
msgfplus_v2018_09_12	msgfplus2csv_py_v1_0_0
msgfplus_v2019_01_22	msgfplus2csv_py_v1_0_0
msgfplus_v2019_04_18	msgfplus2csv_py_v1_0_0
msgfplus_v2019_07_03	msgfplus2csv_py_v1_0_0
msgfplus_v9979	msgfplus2csv_py_v1_0_0

5.1.196 msgfplus_protocol_id

MS-GF+ specific protocol identifier. Protocols are used to enable scoring parameters for enriched and/or labeled samples.

Default value 0

type select

triggers rerun True

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30

- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979

Ursgal key translations for *msgfplus_protocol_id*

Style	Translation
msgfplus_style_1	-protocol

Ursgal value translations

Ursgal Value	Translated Value
.	msgfplus_style_1
0	0
1	1
2	2
3	3

5.1.197 myrimatch_class_size_multiplier

Myrimatch ClassSizeMultiplier

Default value 2

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_class_size_multiplier*

Style	Translation
myrimatch_style_1	ClassSizeMultiplier

5.1.198 myrimatch_num_int_classes

Myrimatch NumIntensityClasses

Default value 3

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_num_int_classes*

Style	Translation
myrimatch_style_1	NumIntensityClasses

5.1.199 myrimatch_num_mz_fidelity_classes

Myrimatch NumMzFidelityClasses

Default value 3

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_num_mz_fidelity_classes*

Style	Translation
myrimatch_style_1	NumMzFidelityClasses

5.1.200 myrimatch_prot_sampl_time

Myrimatch ProteinSamplingTime

Default value 15

type int

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_prot_sampl_time*

Style	Translation
myrimatch_style_1	ProteinSamplingTime

5.1.201 myrimatch_smart_plus_three

Use Myrimatch UseSmartPlusThreeModel

Default value True

type bool

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_smart_plus_three*

Style	Translation
myrimatch_style_1	UseSmartPlusThreeModel

Ursgal value translations

Ursgal Value	Translated Value
.	myrimatch_style_1
0	0
1	1

5.1.202 myrimatch_tic_cutoff

Myrimatch TicCutoffPercentage

Default value 0.98

type float

triggers rerun True

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal key translations for *myrimatch_tic_cutoff*

Style	Translation
myrimatch_style_1	TicCutoffPercentage

5.1.203 mz_score_percentile

weighting factor for pyQms mz score

Default value 0.4

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *mz_score_percentile*

Style	Translation
pyqms_style_1	MZ_SCORE_PERCENTILE
sugarpy_plot_style_1	MZ_SCORE_PERCENTILE
sugarpy_run_style_1	MZ_SCORE_PERCENTILE

5.1.204 mz_transformation_factor

Factor which will be multiplied with mz before conversion to integer

Default value 1000

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *mz_transformation_factor*

Style	Translation
pyqms_style_1	MZ_TRANSFORMATION_FACTOR
sugarpy_plot_style_1	MZ_TRANSFORMATION_FACTOR
sugarpy_run_style_1	MZ_TRANSFORMATION_FACTOR

5.1.205 mzidentml_compress

Compress mzidentml_lib output files

Default value False

type bool

triggers rerun True

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal key translations for *mzidentml_compress*

Style	Translation
mzidentml_style_1	-compress

Ursgal value translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

5.1.206 mzidentml_converter_version

mzidentml converter version: version name

Default value mzidentml_lib_1_6_10

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *mzidentml_converter_version*

Style	Translation
ucontroller_style_1	mzidentml_converter_version

5.1.207 mzidentml_export_type

Defines which paramters shoul be exporte by mzidentml_lib

Default value exportPSMs

type select

triggers rerun True

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal key translations for *mzidentml_export_type*

Style	Translation
mzidentml_style_1	-exportType

5.1.208 mzidentml_function

Defines the mzidentml_lib function to be used. Note: only 'Mzid2Csv' is supported so far

Default value Mzid2Csv

type select

triggers rerun True

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal key translations for *mzidentml_function*

Style	Translation
mzidentml_style_1	mzidentml_function

5.1.209 mzidentml_output_fragmentation

Include fragmentation in mzidentml_lib output

Default value False

type bool

triggers rerun True

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal key translations for *mzidentml_output_fragmentation*

Style	Translation
mzidentml_style_1	-outputFragmentation

Ursgal value translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

5.1.210 mzidentml_verbose_output

Verbose mzidentml_lib output

Default value False

type bool

triggers rerun True

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal key translations for *mzidentml_verbose_output*

Style	Translation
mzidentml_style_1	-verboseOutput

Ursgal value translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

5.1.211 mzml2mgf_converter_version

mzml to mgf converter version: version name

Default value mzml2mgf_2_0_0

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *mzml2mgf_converter_version*

Style	Translation
ucontroller_style_1	mzml2mgf_converter_version

5.1.212 mzml_input_files

List of paths to the mzML input files

Default value None

type list

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- glycopeptide_fragmentor_1_0_0
- ptmshepherd_0_3_5

Ursgal key translations for *mzml_input_files*

Style	Translation
glycopeptide_fragmentor_style_1	mzml_file_list
ptmshepherd_style_1	dataset
sugarpy_plot_style_1	mzml_file
sugarpy_run_style_1	mzml_file

5.1.213 neutral_loss_enabled

Neutral losses enabled for spectrum algorithm: set True or False

Default value False

type bool

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *neutral_loss_enabled*

Style	Translation
xtandem_style_1	spectrum, use neutral loss window

Ursgal value translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

5.1.214 neutral_loss_mass

Sets the centre of the window for ignoring neutral molecule losses.

Default value 0

type int

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *neutral_loss_mass*

Style	Translation
xtandem_style_1	spectrum, neutral loss mass

5.1.215 neutral_loss_window

Neutral loss window: sets the width of the window for ignoring neutral molecule losses.

Default value 0

type int

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *neutral_loss_window*

Style	Translation
xtandem_style_1	spectrum, neutral loss window

5.1.216 noise_suppression_enabled

Used noise suppresssion

Default value False

type bool

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer

Ursgal key translations for *noise_suppression_enabled*

Style	Translation
xtandem_style_1	spectrum, use noise suppression

Ursgal value translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

5.1.217 normalize_intensities

normalize intensity results

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *normalize_intensities*

Style	Translation
flash_lfq_style_1	–nor

5.1.218 num_bins_for_smoothing

Number of bins on each side of a bin that the weight of the bin is smoothed across. This smoothing traces a normal distribution.

Default value 3

type int

triggers rerun True

Available in unodes

- ptmshepherd_0_3_5

Ursgal key translations for *num_bins_for_smoothing*

Style	Translation
ptmshepherd_style_1	histo_smoothbins

5.1.219 num_compared_psms

Maximum number of PSMs (sorted by score, starting with the best scoring PSM) that are compared

Default value 2

type int

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *num_compared_psms*

Style	Translation
sanitize_csv_style_1	num_compared_psms

5.1.220 num_i_decimals

Number of decimals for intensity (peak)

Default value 7

type int

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0

Ursgal key translations for *num_i_decimals*

Style	Translation
mzml2mgf_style_1	number_of_i_decimals

5.1.221 num_match_spec

Maximum number of peptide spectrum matches to report for each spectrum

Default value 10

type int

triggers rerun True

Available in unodes

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03

- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- pepnovo_3_1
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pnovo_3_1_3

Ursgal key translations for *num_match_spec*

Style	Translation
msamanda_style_1	max_rank
msfragger_style_1	output_report_topN
msfragger_style_2	output_report_topN
msfragger_style_3	output_report_topN
msgfplus_style_1	-n
myrimatch_style_1	MaxResultRank
omssa_style_1	-hc
pepnovo_style_1	-num_solutions
pnovo_style_1	report_pep

5.1.222 num_mz_decimals

Number of decimals for m/z mass

Default value 7

type int

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0

Ursgal key translations for *num_mz_decimals*

Style	Translation
mzml2mgf_style_1	number_of_mz_decimals

5.1.223 omssa_cp

Omssa: eliminate charge reduced precursors in spectra

Default value False

type bool

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_cp*

Style	Translation
omssa_style_1	-cp

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
0	0
1	1

5.1.224 omssa_h1

Omssa: number of peaks allowed in single charge window

Default value 2

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_h1*

Style	Translation
omssa_style_1	-h1

5.1.225 omssa_h2

Omssa: number of peaks allowed in double charge window

Default value 2

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_h2*

Style	Translation
omssa_style_1	-h2

5.1.226 omssa_ht

Omssa: number of m/z values corresponding to the most intense peaks that must include one match to the theoretical peptide

Default value 6

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_ht*

Style	Translation
omssa_style_1	-ht

5.1.227 omssa_mm

Omssa: the maximum number of mass ladders to generate per database peptide

Default value 128

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_mm*

Style	Translation
omssa_style_1	-mm

5.1.228 omssa_ta

Omssa: automatic mass tolerance adjustment fraction

Default value 1.0

type float

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_ta*

Style	Translation
omssa_style_1	-ta

5.1.229 omssa_tex

Omssa: threshold in Da above which the mass of neutron should be added in exact mass search

Default value 1446.94

type float

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_tex*

Style	Translation
omssa_style_1	-tex

5.1.230 omssa_verbose

Omssa: verbose info print

Default value False

type bool

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_verbose*

Style	Translation
omssa_style_1	-ni

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
0	
1	-ni

5.1.231 omssa_w1

Omssa: single charge window in Da

Default value 27

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_w1*

Style	Translation
omssa_style_1	-w1

5.1.232 omssa_w2

Omssa: double charge window in Da

Default value 14

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_w2*

Style	Translation
omssa_style_1	-w2

5.1.233 omssa_z1

Omssa: fraction of peaks below precursor used to determine if spectrum is charge 1

Default value 0.95

type float

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_z1*

Style	Translation
omssa_style_1	-z1

5.1.234 omssa_zc

Should charge plus one be determined algorithmically?

Default value True

type bool

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_zc*

Style	Translation
omssa_style_1	-zc

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
0	0
1	1

5.1.235 omssa_zcc

Omssa: how should precursor charges be determined?, use a range

Default value 2

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_zcc*

Style	Translation
omssa_style_1	-zcc

5.1.236 omssa_zt

Minimum precursor charge to start considering multiply charged products

Default value 3

type int

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *omssa_zt*

Style	Translation
omssa_style_1	-zt

5.1.237 only_precursor_charge

use only precursor charge state

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *only_precursor_charge*

Style	Translation
flash_lfq_style_1	–chg

5.1.238 output_aa_probs

Output probabilities for each amino acid.

Default value True

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *output_aa_probs*

Style	Translation
pepnovo_style_1	-output_aa_probs

5.1.239 output_add_features

Add features to the output of MSGF+

Default value True

type bool

triggers rerun True

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979

Ursgal key translations for *output_add_features*

Style	Translation
msgfplus_style_1	-addFeatures

Ursgal value translations

Ursgal Value	Translated Value
.	msgfplus_style_1
0	0
1	1

5.1.240 output_cum_probs

Output cumulative probabilities.

Default value True

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *output_cum_probs*

Style	Translation
pepnovo_style_1	-output_cum_probs

5.1.241 output_file_incl_path

Path to output file 'None' : None

Default value None

type str

triggers rerun True

Available in unodes

- generate_target_decoy_1_0_0
- merge_csvs_1_0_0
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16

- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0
- qvality_2_02
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- venndiagram_1_0_0
- venndiagram_1_1_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- ptminer_1_0

Ursgal key translations for *output_file_incl_path*

Style	Translation
generate_target_decoy_style_1	output_file
merge_csvs_style_1	output
moda_style_1	-o
msamanda_style_1	output_file_incl_path
msgfplus_style_1	-o
myrimatch_style_1	output_file_incl_path
mzidentml_style_1	output_file_incl_path
novor_style_1	output_file_incl_path
omssa_style_1	output_file_incl_path
pepnovo_style_1	output_file_incl_path
percolator_style_1	output_file_incl_path
ptminer_style_1	output_file_incl_path
qvality_style_1	-o
sugarpy_plot_style_1	output_file
sugarpy_run_style_1	output_file
venndiagram_style_1	output_file
xtandem_style_1	output, path

5.1.242 output_file_type

Output file type. If set to 'default', default output file tzpes for each engine are used. Note: not every file type is supported by every engine and usin non-default types might cause problems during conversion to .csv.

Default value default

type select

triggers rerun True

Available in unodes

- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- thermo_raw_file_parser_1_1_2
- tag_graph_1_8_0

Ursgal key translations for *output_file_type*

Style	Translation
omssa_style_1	('-oc', '-ox')
tag_graph_style_1	generatePepXML
thermo_raw_file_parser_style_1	-f
xtandem_style_1	output, mzid

Ursgal value translations

Ursgal Value	Translated Value			
.	omssa_style_1	tag_graph_style_1	thermo_raw_file_parser_style	xtandem_style_1
.csv	-oc	n/t	n/t	n/t
.mgf	n/t	n/t	0	n/t
.mzid	n/t	n/t	n/t	yes
.mzml	n/t	n/t	1	n/t
.omx	-ox	n/t	n/t	n/t
.pepXML	n/t	1	n/t	n/t
default	-oc	0	1	no
indexed_mzml	n/t	n/t	2	n/t
parquet	n/t	n/t	3	n/t

5.1.243 output_prm

Only print spectrum graph nodes with scores.

Default value False

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *output_prm*

Style	Translation
pepnovo_style_1	-prm

5.1.244 output_prm_norm

Prints spectrum graph scores after normalization and removal of negative scores.

Default value False

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *output_prm_norm*

Style	Translation
pepnovo_style_1	-prm_norm

5.1.245 output_q_values

Output Q-values

Default value True

type bool

triggers rerun True

Available in unodes

- msgfplus2csv_v2016_09_16
- msgfplus2csv_v2017_01_27

Ursgal key translations for *output_q_values*

Style	Translation
msgfplus_style_1	-showQValue

Ursgal value translations

Ursgal Value	Translated Value
.	msgfplus_style_1
0	0
1	1

5.1.246 peak_colors

The dict defines the colors of “matched”, “unmatched”, “raw” peaks and “labels”

Default value [‘labels’, ‘matched’, ‘raw’, ‘unmatched’]

type dict

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *peak_colors*

Style	Translation
sugarpy_plot_style_1	peak_colors

5.1.247 pepnovo_tag_length

Returns peptide sequences of the specified length (only lengths 3-6 are allowed) 0 : None

Default value None

type int

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *pepnovo_tag_length*

Style	Translation
pepnovo_style_1	-tag_length

5.1.248 peptide_mapper_class_version

version 3 and 4 are the fastest and most memory efficient class versions, version 2 is the classic approach

Default value UPeptideMapper_v4

type str

triggers rerun True

Available in unodes

- upeptide_mapper_1_0_0

Ursgal key translations for *peptide_mapper_class_version*

Style	Translation
upeptide_mapper_style_1	peptide_mapper_class_version

5.1.249 peptide_mapper_converter_version

determines which upeptide mapper node should be used

Default value upeptide_mapper_1_0_0

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *peptide_mapper_converter_version*

Style	Translation
ucontroller_style_1	peptide_mapper_converter_version

5.1.250 percolator_post_processing

Method to assign FDR and PEP to PSMs

Default value tdc

type select

triggers rerun True

Available in unodes

- percolator_3_2_1
- percolator_3_4_0

Ursgal key translations for *percolator_post_processing*

Style	Translation
percolator_style_1	('-y', '-Y')

5.1.251 pipi_mz_bin_offset

PIPI mz_bin_offset

Default value 0.0

type float

triggers rerun True

Available in unodes

- pipi_1_4_5
- pipi_1_4_6

Ursgal key translations for *pipi_mz_bin_offset*

Style	Translation
pipi_style_1	mz_bin_offset

5.1.252 plotly_layout

The dict defines plotly layout options. Checkout https://plot.ly/python/reference/#layout for all available options

Default value []

type dict

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *plotly_layout*

Style	Translation
sugarpy_plot_style_1	plotly_layout

5.1.253 pparse_options

Dictionary to specify options and their value for pParse. For available options see http://pfind.ict.ac.cn/software/pParse/#

Default value ['-F', '-m', '-p']

type dict

triggers rerun True

Available in unodes

- pparse_2_0
- pparse_2_2_1

Ursgal key translations for *pparse_options*

Style	Translation
pparse_style_1	pparse_options

5.1.254 precursor_charge_dependency

charge dependency of precursor mass tolerance (none or linear)

Default value linear

type select

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *precursor_charge_dependency*

Style	Translation
omssa_style_1	-tez

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
linear	1
none	0

5.1.255 precursor_isotope_range

Error range for incorrect carbon isotope parent ion assignment

Default value 0,1

type select

triggers rerun True

Available in unodes

- kojak_1_5_3
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- pepnovo_3_1
- unify_csv_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- ptmshepherd_0_3_5

Ursgal key translations for *precursor_isotope_range*

Style	Translation
kojak_style_1	precursor_isotope_range
msfragger_style_1	isotope_error
msfragger_style_2	isotope_error
msfragger_style_3	isotope_error
msgfplus_style_1	-ti
myrimatch_style_1	MonoisotopeAdjustmentSet
omssa_style_1	-ti
pepnovo_style_1	-correct_pm
ptmshepherd_style_1	isotope_error
unify_csv_style_1	precursor_isotope_range
xtandem_style_1	spectrum, parent monoisotopic mass isotope error

Ursgal value translations

Ursgal Value	Translated Value							
.	kojak_style_1	msfragger_style_1	msfragger_style_2	msfragger_style_3	myrimatch_style_1	omssa_style_1	ptmshepherd_style_1	xtandem_style_1
0	0	0	0	0	[0,]	0	0	no
0,1	1	0/1	0/1	0/1	[0,1]	1	0/1	yes
0,1,2	n/t	n/t	n/t	n/t	[0,1,2]	n/t	n/t	n/t
0,2	2	0/1/2	0/1/2	0/1/2	n/t	2	0/1/2	yes

5.1.256 precursor_mass_tolerance_minus

Lower precursor mass tolerance; maximum negative deviation of measured from calculated parent ion mass.

Default value 5

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299

- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- unify_csv_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3

- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- deepnovo_0_0_1
- deepnovo_pointnovo
- pnovo_3_1_3
- ptmshepherd_0_3_5

Ursgal key translations for *precursor_mass_tolerance_minus*

Style	Translation
deepnovo_style_1	('precursor_mass_tolerance', 'precursor_mass_ppm')
kojak_style_1	ppm_tolerance_pre
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol
msfragger_style_1	precursor_mass_lower
msfragger_style_2	precursor_mass_lower
msfragger_style_3	precursor_mass_lower
msgfplus_style_1	-t
myrimatch_style_1	MonoPrecursorMzTolerance
novor_style_1	precursorErrorTol
omssa_style_1	-te
pepnovo_style_1	-pm_tolerance
pglyco_db_style_1	search_precursor_tolerance
pipi_style_1	ms1_tolerance
pnovo_style_1	pep_tol
ptminer_style_1	precursor_tol
ptmshepherd_style_1	precursor_tol
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
unify_csv_style_1	precursor_mass_tolerance_minus
xtandem_style_1	spectrum, parent monoisotopic mass error minus

5.1.257 precursor_mass_tolerance_plus

Upper precursor mass tolerance; maximum positive deviation of measured from calculated parent ion mass.

Default value 5

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- unify_csv_1_0_0

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- deepnovo_0_0_1
- deepnovo_pointnovo
- pnovo_3_1_3
- flash_lfq_1_1_1
- ptmshepherd_0_3_5

Ursgal key translations for *precursor_mass_tolerance_plus*

Style	Translation
deepnovo_style_1	('precursor_mass_tolerance', 'precursor_mass_ppm')
flash_lfq_style_1	–ppm
kojak_style_1	ppm_tolerance_pre
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol
msfragger_style_1	precursor_mass_upper
msfragger_style_2	precursor_mass_upper
msfragger_style_3	precursor_mass_upper
msgfplus_style_1	-t
myrimatch_style_1	MonoPrecursorMzTolerance
novor_style_1	precursorErrorTol
omssa_style_1	-te
pepnovo_style_1	-pm_tolerance
pglyco_db_style_1	search_precursor_tolerance
pipi_style_1	ms1_tolerance
pnovo_style_1	pep_tol
ptminer_style_1	precursor_tol
ptmshepherd_style_1	precursor_tol
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
unify_csv_style_1	precursor_mass_tolerance_minus
xtandem_style_1	spectrum, parent monoisotopic mass error plus

5.1.258 precursor_mass_tolerance_unit

Precursor mass tolerance unit: available in ppm (parts-per-millon), da (Dalton) or mmu (Milli mass unit)

Default value ppm

type select

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504

- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- novor_1_05
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pipi_1_4_5
- pipi_1_4_6
- pyqms_1_0_0

- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- pglyco_db_2_2_0
- ptminer_1_0
- pglyco_db_2_2_2
- deepnovo_0_0_1
- deepnovo_pointnovo
- pnovo_3_1_3
- ptmshepherd_0_3_5

Ursgal key translations for *precursor_mass_tolerance_unit*

Style	Translation
deepnovo_style_1	(‘precursor_mass_tolerance’, ‘precursor_mass_ppm’)
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol unit
msfragger_style_1	precursor_mass_units
msfragger_style_2	precursor_mass_units
msfragger_style_3	precursor_mass_units
msgfplus_style_1	-t
myrimatch_style_1	MonoPrecursorMzTolerance
novor_style_1	precursorErrorTol
omssa_style_1	-teppm
pepnovo_style_1	precursor_mass_tolerance_unit
pglyco_db_style_1	search_precursor_tolerance_type
pipi_style_1	ms1_tolerance_unit
pnovo_style_1	pep_tol_type_ppm
ptminer_style_1	precursor_tol_type
ptmshepherd_style_1	precursor_mass_units
pyqms_style_1	REL_MZ_RANGE
sugarpy_plot_style_1	REL_MZ_RANGE
sugarpy_run_style_1	REL_MZ_RANGE
xtandem_style_1	spectrum, parent monoisotopic mass error units

Ursgal value translations

Ursgal Value	Translated Value													
.	msamanda_style_1	moda_style_1	msfragger_style_1	msfragger_style_2	msgfplus_style_1	myrimatch_style_1	novor_style_1	omssa_style_1	pglyco_db_style_1	pipi_style_1	pnovo_style_1	ptminer_style_1	ptmshepherd_style_1	xtandem_style_1
da	Da	0	0	0	Da	Da	Da		Da	0	0	0	0	Daltons
ppm	n/t	1	1	1	n/t	n/t	n/t	-teppm	n/t	1	1	1	1	n/t

5.1.259 precursor_mass_type

Precursor mass type: monoisotopic or average

Default value monoisotopic

type select

triggers rerun True

Available in unodes

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9

Ursgal key translations for *precursor_mass_type*

Style	Translation
msamanda_style_1	monoisotopic
myrimatch_style_1	PrecursorMzToleranceRule
omssa_style_1	-tem

Ursgal value translations

Ursgal Value	Translated Value		
.	msamanda_style_1	myrimatch_style_1	omssa_style_1
average	false	average	1
monoisotopic	true	mono	0

5.1.260 precursor_max_charge

Maximal accepted parent ion charge

Default value 5

type int

triggers rerun True

Available in unodes

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- mzml2mgf_2_0_0
- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628

- msfragger_2_3
- msfragger_3_0
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *precursor_max_charge*

Style	Translation
msamanda_style_1	considered_charges
msfragger_style_1	precursor_max_charge
msfragger_style_2	precursor_max_charge
msfragger_style_3	precursor_max_charge
msgfplus_style_1	-maxCharge
myrimatch_style_1	NumChargeStates
mzml2mgf_style_1	precursor_max_charge
omssa_style_1	-zh
pyqms_style_1	precursor_max_charge
sugarpy_plot_style_1	max_charge
sugarpy_run_style_1	max_charge

5.1.261 precursor_max_mass

Maximal parent ion mass in Da. Adjusted to default used by MSFragger

Default value 7000

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- myrimatch_2_1_138
- myrimatch_2_2_140
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pglyco_db_2_2_0
- pglyco_db_2_2_2

- pnovo_3_1_3
- deepnovo_pointnovo

Ursgal key translations for *precursor_max_mass*

Style	Translation
deepnovo_style_1	MZ_MAX
kojak_style_1	precursor_max_mass
msfragger_style_1	precursor_max_mass
msfragger_style_2	precursor_max_mass
msfragger_style_3	precursor_max_mass
myrimatch_style_1	MaxPeptideMass
pglyco_db_style_1	max_peptide_weight
pnovo_style_1	mass_upper_bound
xtandem_style_1	spectrum, minimum parent m+h

5.1.262 precursor_min_charge

Minimal accepted parent ion charge

Default value 1

type int

triggers rerun True

Available in unodes

- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28

- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- mzml2mgf_2_0_0
- omssa_2_1_9
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *precursor_min_charge*

Style	Translation
msamanda_style_1	considered_charges
msfragger_style_1	precursor_min_charge
msfragger_style_2	precursor_min_charge
msfragger_style_3	precursor_min_charge
msgfplus_style_1	-minCharge
mzml2mgf_style_1	precursor_min_charge
omssa_style_1	-zl
pyqms_style_1	precursor_min_charge
sugarpy_plot_style_1	min_charge
sugarpy_run_style_1	min_charge

5.1.263 precursor_min_mass

Minimal parent ion mass

Default value 400

type int

triggers rerun True

Available in unodes

- kojak_1_5_3
- myrimatch_2_1_138
- myrimatch_2_2_140
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0
- pglyco_db_2_2_0
- pglyco_db_2_2_2
- pnovo_3_1_3

Ursgal key translations for *precursor_min_mass*

Style	Translation
kojak_style_1	precursor_min_mass
msfragger_style_1	precursor_min_mass
msfragger_style_2	precursor_min_mass
msfragger_style_3	precursor_min_mass
myrimatch_style_1	MinPeptideMass
pglyco_db_style_1	min_peptide_weight
pnovo_style_1	mass_lower_bound
xtandem_style_1	spectrum, minimum parent m+h

5.1.264 precursor_true_tolerance

True precursor mass tolerance (window is +/- this value). Used for tie breaker of results (in spectrally ambiguous cases) and zero bin boosting in open searches (0 disables these features). This option is STRONGLY recommended for open searches.

Default value 5

type int

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *precursor_true_tolerance*

Style	Translation
msfragger_style_1	precursor_true_tolerance
msfragger_style_2	precursor_true_tolerance
msfragger_style_3	precursor_true_tolerance

5.1.265 precursor_true_units

Mass tolerance units fo precursor_true_tolerance

Default value ppm

type str

triggers rerun True

Available in unodes

- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *precursor_true_units*

Style	Translation
msfragger_style_1	precursor_true_units
msfragger_style_2	precursor_true_units
msfragger_style_3	precursor_true_units

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
da	0	0	0
ppm	1	1	1

5.1.266 preferred_engines

List of engines that should be preferred when sanitizing results. In combination with "accept_conflicting_psms"=True, conflicting PSMs for only the preferred engine are selected. If no PSMs for that engine exist, the second preferred engine is used, and so on

Default value []

type list

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *preferred_engines*

Style	Translation
sanitize_csv_style_1	preferred_engines

5.1.267 prefix

'None' : None

Default value None

type None

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *prefix*

Style	Translation
ucontroller_style_1	prefix

5.1.268 protein_delimiter

This delimiter seperates protein IDs/names in the unified csv

Default value <|>

type str

triggers rerun True

Available in unodes

- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0
- unify_csv_1_0_0
- upeptide_mapper_1_0_0

Ursgal key translations for *protein_delimiter*

Style	Translation
percolator_style_1	protein_delimiter
unify_csv_style_1	protein_delimiter
upeptide_mapper_style_1	protein_delimiter

5.1.269 psm_colnames_to_merge_multiple_values

Defines the column names which should have their different values merged into a single value when merging rows correspondin
min_value most_frequent avg_value

Default value []

type dict

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *psm_colnames_to_merge_multiple_values*

Style	Translation
ucontroller_style_1	colnames_to_merge_multiple_values

5.1.270 psm_defining_colnames

List of column names that are used to define unique PSMs and to merge multiple lines of the same PSM (if specified). The validation_score_field is automatically added to this list.

Default value ['Spectrum Title', 'Sequence', 'Modifications', 'Mass Difference', 'Charge', 'Is decoy']

type list

triggers rerun True

Available in unodes

- ucontroller
- sanitize_csv_1_0_0
- combine_pep_1_0_0
- unify_csv_1_0_0

Ursgal key translations for *psm_defining_colnames*

Style	Translation
combine_pep_style_1	columns_for_grouping
sanitize_csv_style_1	psm_defining_colnames
ucontroller_style_1	psm_defining_colnames
unify_csv_style_1	psm_defining_colnames

5.1.271 psm_merge_delimiter

This delimiter seperates differing values for merged rows in the unified csv

Default value ;

type str

triggers rerun True

Available in unodes

- unify_csv_1_0_0
- ucontroller

Ursgal key translations for *psm_merge_delimiter*

Style	Translation
ucontroller_style_1	psm_merge_delimiter
unify_csv_style_1	psm_merge_delimiter

5.1.272 ptmshepherd_localization_background

Background of peptides against which localization enrichment scores are calculated

Default value peptides_entire_dataset

type select

triggers rerun True

Available in unodes

- ptmshepherd_0_3_5

Ursgal key translations for *ptmshepherd_localization_background*

Style	Translation
ptmshepherd_style_1	localization_background

Ursgal value translations

Ursgal Value	Translated Value
.	ptmshepherd_style_1
peptides_entire_dataset	3
peptides_same_bin	1
psms_entire_dataset	3
psms_same_bin	2

5.1.273 ptmshepherd_peak_picking_params

Dictionary to specify options for peak picking for PTM-Shepherd.

Default value ['peakpicking_mass_units', 'peakpicking_promRatio', 'peakpicking_topN', 'peakpicking_width']

type dict

triggers rerun True

Available in unodes

- ptmshepherd_0_3_5

Ursgal key translations for *ptmshepherd_peak_picking_params*

Style	Translation
ptmshepherd_style_1	('peakpicking_promRatio', 'peakpicking_mass_units', 'peakpicking_width', 'peakpicking_topN')

5.1.274 pymzml_spec_id_attribute

Specify the spectrum ID attribute to be used to access the spectrum ID (ID, id_dict or index). Given as a dict (key = attribute, value = key in id_dict). For .wiff files, during conversion to mzML, spectrum IDs are formatted differently; pymzml can deal with this by returning an id_dict or accessing the index.

Default value ['ID']

type dict

triggers rerun True

Available in unodes

- mzml2mgf_2_0_0

Ursgal key translations for *pymzml_spec_id_attribute*

Style	Translation
mzml2mgf_style_1	spec_id_attribute

5.1.275 pyqms_min_rel_isotope_peak_intensity

Defines the relative minimum peak intensity within an isotopologue to be considered for matching

Default value 0.01

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *pyqms_min_rel_isotope_peak_intensity*

Style	Translation
pyqms_style_1	MIN_REL_PEAK_INTENSITY_FOR_MATCHING
sugarpy_plot_style_1	MIN_REL_PEAK_INTENSITY_FOR_MATCHING
sugarpy_run_style_1	MIN_REL_PEAK_INTENSITY_FOR_MATCHING

5.1.276 pyqms_trivial_names

Trivial name lookup mapping molecules to a trivial name

Default value None

type dict

triggers rerun True

Available in unodes

- pyqms_1_0_0

Ursgal key translations for *pyqms_trivial_names*

Style	Translation
pyqms_style_1	trivial_names

5.1.277 pyqms_verbosity

verbosity for pyqms

Default value True

type bool

triggers rerun False

Available in unodes

- pyqms_1_0_0

Ursgal key translations for *pyqms_verbosity*

Style	Translation
pyqms_style_1	pyqms_verbosity

5.1.278 quantification_evidences

Molecules to quantify. Can be either a list of strings or a csv file

Default value None

type list

triggers rerun True

Available in unodes

- pyqms_1_0_0
- flash_lfq_1_1_1

Ursgal key translations for *quantification_evidences*

Style	Translation
flash_lfq_style_1	evidences
pyqms_style_1	evidences

5.1.279 qvality_cross_validation

The relative crossvalidation step size used as treshhold before ending the iterations, qvality determines step size automatically when set to 0

Default value 0

type int

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *qvality_cross_validation*

Style	Translation
qvality_style_1	-c

5.1.280 qvality_epsilon_step

The relative step size used as treshhold before cross validation error is calculated, qvality determines step size automatically when set to 0

Default value 0

type int

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *qvality_epsilon_step*

Style	Translation
qvality_style_1	-s

5.1.281 qvality_number_of_bins

Number of bins used in qvality

Default value 500

type int

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *qvality_number_of_bins*

Style	Translation
qvality_style_1	-n

5.1.282 qvality_verbose

Verbose qvality output (range from 0 = no processing info to 5 = all)

Default value 2

type select

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *qvality_verbose*

Style	Translation
qvality_style_1	-v

Ursgal value translations

Ursgal Value	Translated Value
.	qvality_style_1
1	1
2	2
3	3
4	4
5	5

5.1.283 random_seed

Random seed for random number generators

Default value 10

type int

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *random_seed*

Style	Translation
flash_lfq_style_1	–rns

5.1.284 raw_ident_csv_suffix

CSV suffix of raw indentification: this is the conversion result after CSV conversion but before adding retention time

Default value .csv

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *raw_ident_csv_suffix*

Style	Translation
ucontroller_style_1	raw_ident_csv_suffix

5.1.285 rel_intensity_error

Relative intensity error range (for the most intense peak)

Default value 0.2

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *rel_intensity_error*

Style	Translation
pyqms_style_1	REL_I_RANGE
sugarpy_plot_style_1	REL_I_RANGE
sugarpy_run_style_1	REL_I_RANGE

5.1.286 remove_precursor_peak

Remove precursor peaks from spectrum. Options are: "off", "remove_precursor_charge" (removes peaks with same charge as precursor), "remove_all_charges" (removes peaks of all charges)

Default value off

type select

triggers rerun True

Available in unodes

- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *remove_precursor_peak*

Style	Translation
msfragger_style_3	remove_precursor_peak

Ursgal value translations

Ursgal Value	Translated Value
.	msfragger_style_3
off	0
remove_all_charges	2
remove_precursor_charge	1

5.1.287 remove_precursor_range

The given mass range is used to remove precursor peaks.

Default value [-1.5, 1.5]

type list

triggers rerun True

Available in unodes

- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *remove_precursor_range*

Style	Translation
msfragger_style_3	remove_precursor_range

5.1.288 remove_redundant_psms

If True, redundant PSMs (e.g. the same identification reported by multiple engines) for the same spectrum are removed. An identification is defined by psm_defining_colnames

Default value True

type bool

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *remove_redundant_psms*

Style	Translation
sanitize_csv_style_1	remove_redundant_psms

5.1.289 remove_temporary_files

Remove temporary files: True or False

Default value True

type bool

triggers rerun True

Available in unodes

- ucontroller
- tag_graph_1_8_0

Ursgal key translations for *remove_temporary_files*

Style	Translation
tag_graph_style_1	('cleanInputDataFilesFromOutput', 'cleanIntermediateFiles')
ucontroller_style_1	remove_temporary_files

5.1.290 require_msms_id

Require MS/MS match in condition to consider quantification

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *require_msms_id*

Style	Translation
flash_lfq_style_1	–rmc

5.1.291 required_percentile_peak_overlap

Minimum percentile overlap for matching labeled peaks

Default value 0.5

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *required_percentile_peak_overlap*

Style	Translation
pyqms_style_1	REQUIRED_PERCENTILE_PEAK_OVERLAP
sugarpy_plot_style_1	REQUIRED_PERCENTILE_PEAK_OVERLAP
sugarpy_run_style_1	REQUIRED_PERCENTILE_PEAK_OVERLAP

5.1.292 rounded_mass_decimals

Masses of modifications are rounded in order to match them to their corresponding unimod name. Use this parameter to set the number of decimal places after rounding.

Default value 3

type int

triggers rerun True

Available in unodes

- unify_csv_1_0_0

Ursgal key translations for *rounded_mass_decimals*

Style	Translation
unify_csv_style_1	rounded_mass_decimals

5.1.293 rt_border_tolerance

Retention time border tolerance (in min) for curating RT windows

Default value 1

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *rt_border_tolerance*

Style	Translation
pyqms_style_1	rt_border_tolerance
sugarpy_plot_style_1	rt_border_tolerance
sugarpy_run_style_1	rt_border_tolerance

5.1.294 rt_pickle_name

name of the pickle that is used to map the retention time

Default value _ursgal_lookup.pkl

type str

triggers rerun True

Available in unodes

- ucontroller
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0
- mgf_to_rt_lookup_1_0_0
- unify_csv_1_0_0

Ursgal key translations for *rt_pickle_name*

Style	Translation
mgf_to_rt_lookup_style_1	rt_pickle_name
sugarpy_plot_style_1	scan_rt_lookup
sugarpy_run_style_1	scan_rt_lookup
ucontroller_style_1	rt_pickle_name
unify_csv_style_1	scan_rt_lookup_path

5.1.295 scan_exclusion_list

Spectra rejected during mzml2mgf conversion

Default value []

type list

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0

Ursgal key translations for *scan_exclusion_list*

Style	Translation
mzml2mgf_style_1	scan_exclusion_list

5.1.296 scan_inclusion_list

Exclusively spectra included during mzml2mgf conversion

Default value None

type list

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0

Ursgal key translations for *scan_inclusion_list*

Style	Translation
mzml2mgf_style_1	scan_inclusion_list

5.1.297 scan_skip_modulo_step

Include only the n-th spectrum during mzml2mgf conversion 1 : None

Default value None

type int

triggers rerun True

Available in unodes

- mzml2mgf_1_0_0
- mzml2mgf_2_0_0

Ursgal key translations for *scan_skip_modulo_step*

Style	Translation
mzml2mgf_style_1	scan_skip_modulo_step

Ursgal value translations

Ursgal Value	Translated Value
.	mzml2mgf_style_1

5.1.298 score_correlation_corr

Use correlation correction to score?

Default value True

type bool

triggers rerun True

Available in unodes

- omssa_2_1_9

Ursgal key translations for *score_correlation_corr*

Style	Translation
omssa_style_1	-scorr

Ursgal value translations

Ursgal Value	Translated Value
.	omssa_style_1
0	1
1	0

5.1.299 score_diff_threshold

Minimum score difference between the best PSM and the first rejected PSM of one spectrum, default: 0.01

Default value 0.01

type float

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *score_diff_threshold*

Style	Translation
sanitize_csv_style_1	score_diff_threshold

5.1.300 score_ion_list

List of ion types that are taken into account by the respective search engine.Availabel ion types: a, b, c, x, y, z, -h2o, -nh3, b1, c_terminal, imm (immonium),int (internal), z+1, z+2, b~ (b+HexNAx), y~ (y+HexNAc), Y

Default value ['b', 'y']

type list

triggers rerun True

Available in unodes

- kojak_1_5_3
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_3_0

Ursgal key translations for *score_ion_list*

Style	Translation
kojak_style_1	(‘ion_series_X’, ‘ion_series_Y’, ‘ion_series_Z’, ‘ion_series_A’, ‘ion_series_B’, ‘ion_series_C’)
msamanda_style_1	series
msfragger_style_3	fragment_ion_series
myrimatch_style_1	FragmentationRule
omssa_style_1	(‘-i’, ‘-sct’, ‘-sb1’)
xtandem_style_1	(‘scoring, x ions’, ‘scoring, y ions’, ‘scoring, z ions’, ‘scoring, a ions’, ‘scoring, b ions’, ‘scoring, c ions’)

5.1.301 search_for_saps

Search for potential single amino acid polymorphisms. ‘True’ might cause problems in the downstream processing of th result files (unify_csv, . . .)

Default value False

type bool

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *search_for_saps*

Style	Translation
xtandem_style_1	protein, saps

Ursgal value translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

5.1.302 semi_enzyme

Allows semi-enzymatic peptide ends

Default value False

type bool

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- msamanda_1_0_0_5242
- msamanda_1_0_0_5243
- msamanda_1_0_0_6299
- msamanda_1_0_0_6300
- msamanda_1_0_0_7503
- msamanda_1_0_0_7504
- msamanda_2_0_0_9706
- msamanda_2_0_0_9695
- msamanda_2_0_0_10695
- msamanda_2_0_0_11219
- msamanda_2_0_0_13723
- msamanda_2_0_0_14665
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v2018_01_30
- msgfplus_v2018_06_28
- msgfplus_v2018_09_12
- msgfplus_v2019_01_22
- msgfplus_v2019_04_18
- msgfplus_v2019_07_03
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- unify_csv_1_0_0
- xtandem_cyclone_2010

- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *semi_enzyme*

Style	Translation
moda_style_1	enzyme_constraint_min_number_termini
msamanda_style_1	enzyme specificity
msfragger_style_1	num_enzyme_termini
msfragger_style_2	num_enzyme_termini
msfragger_style_3	num_enzyme_termini
msgfplus_style_1	-ntt
myrimatch_style_1	MinTerminiCleavages
omssa_style_1	semi_enzyme
unify_csv_style_1	semi_enzyme
xtandem_style_1	protein, cleavage semi

Ursgal value translations

Ursgal Value	Translated Value							
.	moda_style_1	msamanda_style_1	msfragger_style_1	msfragger_style_2	msfragger_style_3	msgfplus_style_1	myrimatch_style_1	xtandem_style_1
0	2	Full	2	2	2	2	2	no
1	1	Semi	1	1	1	1	1	yes

5.1.303 show_unodes_in_development

Show ursgal nodes that are in development: False or True

Default value False

type bool

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *show_unodes_in_development*

Style	Translation
ucontroller_style_1	show_unodes_in_development

5.1.304 signal_to_noise_threshold

Only peaks above the given signal to noise (S/N) threshold will be accepted

Default value 0.0

type float

triggers rerun True

Available in unodes

- mzml2mgf_2_0_0

Ursgal key translations for *signal_to_noise_threshold*

Style	Translation
mzml2mgf_style_1	signal_to_noise_threshold

5.1.305 silac_aas_locked_in_experiment

AA which are always SILAC labeled and not considered for calculating partially labeling percentile

Default value None

type list

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *silac_aas_locked_in_experiment*

Style	Translation
pyqms_style_1	SILAC_AAS_LOCKED_IN_EXPERIMENT
sugarpy_plot_style_1	SILAC_AAS_LOCKED_IN_EXPERIMENT
sugarpy_run_style_1	SILAC_AAS_LOCKED_IN_EXPERIMENT

5.1.306 spec_dynamic_range

Internal normalization for MS/MS spectrum: The highest peak (intensity) within a spectrum is set to given value and all other peaks are normalized to this peak. If the normalized value is less than 1 the peak is rejected.

Default value 100

type int

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *spec_dynamic_range*

Style	Translation
xtandem_style_1	spectrum, dynamic range

5.1.307 ssl_score_column_name

Name of the column that includes the scores that should be used for the .ssl file

Default value q-value

type str

triggers rerun True

Available in unodes

- csv2ssl_1_0_0

Ursgal key translations for *ssl_score_column_name*

Style	Translation
csv2ssl_style_1	score_column_name

5.1.308 ssl_score_type

Type of scores used for the .ssl file

Default value PERCOLATOR QVALUE

type select

triggers rerun True

Available in unodes

- csv2ssl_1_0_0

Ursgal key translations for *ssl_score_type*

Style	Translation
csv2ssl_style_1	score_type

5.1.309 sugarpy_decoy_glycan

Glycan (given in the SugarPy Hill noation format) that will be used for matching glycopeptide fragment ions in MS2 spectra from non-glycosylated peptides

Default value End(HexNAc)Hex(5)HexNAc(3)NeuAc(1)dHex(1)

type str

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0
- glycopeptide_fragmentor_1_0_0

Ursgal key translations for *sugarpy_decoy_glycan*

Style	Translation
glycopeptide_fragmentor_style_1	decoy_glycan
sugarpy_plot_style_1	decoy_glycan

5.1.310 sugarpy_include_subtrees

Defines if/how subtrees should be taken into account for plotting molecule elution profiles in SugarPy. Available are: “no_subtrees”, “sum_subtrees”, “individual_subtrees”

Default value no_subtrees

type select

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_include_subtrees*

Style	Translation
sugarpy_plot_style_1	include_subtrees

5.1.311 sugarpy_plot_molecule_dict

The dict contains all peptidoforms (Peptide#Unimod:Pos) as keys and a dict with the glycans (keys) and {‘charges’:set(), ‘file_names’:set()} (value) as values. It can be auto generated from a SugarPy results .csv (use uparam sugarpy_results_file to specify). Don’t use sugarpy_results_file and sugarpy_plot_molecule_dict at the same time!

Default value []

type dict

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_plot_molecule_dict*

Style	Translation
sugarpy_plot_style_1	plot_molecule_dict

5.1.312 sugarpy_plot_peak_types

List of peak types that should be plotted by the SugarPy plot spectrum function. Available are: “matched” (peaks matched by pyQms), “unmatched” (unmatched peaks from matched formulas), “labels@ (for monoisotopic peaks)

Default value [‘matched’, ‘unmatched’, ‘labels’]

type list

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_plot_peak_types*

Style	Translation
sugarpy_plot_style_1	plot_peak_types

5.1.313 sugarpy_plot_types

List of plot types that should be created by the SugarPy plotting function. Available are: "plot_molecule_elution_profile", "plot_glycan_elution_profile", "plot_annotated_spectra", "check_peak_presence", "check_frag_specs"

Default value ['plot_glycan_elution_profile']

type list

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_plot_types*

Style	Translation
sugarpy_plot_style_1	plot_types

5.1.314 sugarpy_remove_subtrees

List of subtree formulas (hill notation) that should not be plotted. Formulas include the complete molecule, i.e. peptide and glycan

Default value []

type list

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_remove_subtrees*

Style	Translation
sugarpy_plot_style_1	remove_subtrees

5.1.315 sugarpy_results_csv

Path to the SugarPy results .csv

Default value None

type str

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_results_csv*

Style	Translation
sugarpy_plot_style_1	result_file

5.1.316 sugarpy_results_pkl

Path to the SugarPy results .pkl

Default value None

type str

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_results_pkl*

Style	Translation
sugarpy_plot_style_1	validated_results_pkl

5.1.317 sugarpy_score_type

Defines the score type used for the y-axis. Available are: “top_scores”, “sum_scores”

Default value top_scores

type select

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *sugarpy_score_type*

Style	Translation
sugarpy_plot_style_1	score_type

5.1.318 svm_c_param

Penalty parameter C of the error term of the post-processing SVM

Default value 1.0

type float

triggers rerun True

Available in unodes

- svm_1_0_0

Ursgal key translations for *svm_c_param*

Style	Translation
svm_style_1	c

5.1.319 tag_graph_config_file

Path to pickled (python-serialized) model configuration file. Use "default" for the default file location in the resources

Default value default

type str

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_config_file*

Style	Translation
tag_graph_style_1	config

5.1.320 tag_graph_fdr_threshold

FDR threshold applied to output by TagGraph

Default value 0.01

type float

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_fdr_threshold*

Style	Translation
tag_graph_style_1	FDRCutoff

5.1.321 tag_graph_init_iterations

Number of iterations in initial EM over all results

Default value 20

type int

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_init_iterations*

Style	Translation
tag_graph_style_1	initIterations

5.1.322 tag_graph_log_em_threshold

p-value threshold applied to output by TagGraph

Default value 2

type int

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_log_em_threshold*

Style	Translation
tag_graph_style_1	logEMCutoff

5.1.323 tag_graph_max_iterations

Maximum number of expectation maximization iterations for FDR assignment

Default value 100

type int

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_max_iterations*

Style	Translation
tag_graph_style_1	maxIterations

5.1.324 tag_graph_model_file

Path to pickled (python-serialized) probabilistic model file. Use "default" for the default file location in the resources

Default value default

type str

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_model_file*

Style	Translation
tag_graph_style_1	model

5.1.325 tag_graph_unimod_file

Path to pickled (python-serialized) unimod dictionary. Use "default" for the default file location in the resources

Default value default

type str

triggers rerun True

Available in unodes

- tag_graph_1_8_0

Ursgal key translations for *tag_graph_unimod_file*

Style	Translation
tag_graph_style_1	unimoddict

5.1.326 test_param1

TEST/DEBUG: Internal Ursgal parameter 1 for debugging and testing.

Default value b

type select

triggers rerun True

Available in unodes

- _test_node

Ursgal key translations for *test_param1*

Style	Translation
_test_node_style_1	test_param1

Ursgal value translations

Ursgal Value	Translated Value
.	_test_node_style_1
a	A
b	B
c	C
d	D
e	E

5.1.327 test_param2

TEST/DEBUG: Internal Ursgal parameter 2 for debugging and testing.

Default value three

type select

triggers rerun True

Available in unodes

- _test_node

Ursgal key translations for *test_param2*

Style	Translation
_test_node_style_1	test_param2

Ursgal value translations

Ursgal Value	Translated Value
.	_test_node_style_1
five	5
four	4
one	1
three	3
two	2

5.1.328 thermo_raw_file_parser_options

Dictionary to specify options and their value for ThermoRawFileParser. If options are given as a flag only, specify 'None' as their value. For available options see https://github.com/compomics/ThermoRawFileParser

Default value ['-e', '-m']

type dict

triggers rerun True

Available in unodes

- thermo_raw_file_parser_1_1_2

Ursgal key translations for *thermo_raw_file_parser_options*

Style	Translation
thermo_raw_file_parser_style_1	('-h', '-m', '-g', '-u', '-k', '-t', '-n', '-v', '-e')

5.1.329 threshold_is_log10

True, if log10 scale has been used for score_diff_threshold.

Default value False

type bool

triggers rerun True

Available in unodes

- sanitize_csv_1_0_0

Ursgal key translations for *threshold_is_log10*

Style	Translation
sanitize_csv_style_1	threshold_is_log10

5.1.330 unify_csv_converter_version

unify csv converter version: version name

Default value unify_csv_1_0_0

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *unify_csv_converter_version*

Style	Translation
ucontroller_style_1	unify_csv_converter_version

5.1.331 upper_mz_limit

Highest considered mz for quantification

Default value 2000

type float

triggers rerun True

Available in unodes

- pyqms_1_0_0
- sugarpy_run_1_0_0
- sugarpy_plot_1_0_0

Ursgal key translations for *upper_mz_limit*

Style	Translation
pyqms_style_1	UPPER_MZ_LIMIT
sugarpy_plot_style_1	UPPER_MZ_LIMIT
sugarpy_run_style_1	UPPER_MZ_LIMIT

5.1.332 ursgal_resource_url

URL that is used to prepare and install resources via corresponding scripts (prepare_resources.py and install_resources.py)

Default value https://www.sas.upenn.edu/~sschulze/ursgal_resources/

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal key translations for *ursgal_resource_url*

Style	Translation
ucontroller_style_1	ursgal_resource_url

5.1.333 ursgal_results_csv

Path to the Ursgal results .csv containing all PSMs in the unified format

Default value None

type str

triggers rerun True

Available in unodes

- sugarpy_plot_1_0_0

Ursgal key translations for *ursgal_results_csv*

Style	Translation
sugarpy_plot_style_1	ursgal_ident_file

5.1.334 use_median_accuracy

Accuracy of identifications (ident_file) are used to calculate the machine_offset_in_ppm. If "all" is selected, the median of all identifications will be used, for "peptide" the median of each peptide will be used.

Default value None

type select

triggers rerun True

Available in unodes

- sugarpy_run_1_0_0

Ursgal key translations for *use_median_accuracy*

Style	Translation
sugarpy_run_style_1	use_median_accuracy

5.1.335 use_prior_probability

specifying whether the prior probability should be used or not

Default value True

type bool

triggers rerun True

Available in unodes

- ptminer_1_0

Ursgal key translations for *use_prior_probability*

Style	Translation
ptminer_style_1	use_prior

Ursgal value translations

Ursgal Value	Translated Value
.	ptminer_style_1
0	0
1	1

5.1.336 use_pyqms_for_mz_calculation

Use pyQms for accurate calculation of isotopologue m/z. This will affect the accuracy (ppm) calculation as well. If True, unify_csv will be significantly slower. Please note that this does not work for any type of labeling yet.

Default value False

type bool

triggers rerun True

Available in unodes

- unify_csv_1_0_0

Ursgal key translations for *use_pyqms_for_mz_calculation*

Style	Translation
unify_csv_style_1	use_pyqms_for_mz_calculation

5.1.337 use_quality_filter

Use filter for low quality spectra.

Default value True

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1

Ursgal key translations for *use_quality_filter*

Style	Translation
pepnovo_style_1	-no_quality_filter

Ursgal value translations

Ursgal Value	Translated Value
.	pepnovo_style_1
0	1
1	0

5.1.338 use_refinement

X! TANDEM can use 'refinement' to improve the speed and accuracy of peptide modelling. This is not included in Ursgal, yet. See further: http://www.thegpm.org/TANDEM/api/refine.html

Default value False

type bool

triggers rerun True

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal key translations for *use_refinement*

Style	Translation
xtandem_style_1	refine

Ursgal value translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

5.1.339 use_shared_peptides

use shared peptides for protein quantification

Default value False

type bool

triggers rerun True

Available in unodes

- flash_lfq_1_1_1

Ursgal key translations for *use_shared_peptides*

Style	Translation
flash_lfq_style_1	–sha

5.1.340 use_spectrum_charge

Does not correct precursor charge.

Default value True

type bool

triggers rerun True

Available in unodes

- pepnovo_3_1
- msfragger_20170103
- msfragger_20171106
- msfragger_20190222
- msfragger_20190628
- msfragger_2_3
- msfragger_3_0

Ursgal key translations for *use_spectrum_charge*

Style	Translation
msfragger_style_1	override_charge
msfragger_style_2	override_charge
msfragger_style_3	override_charge
pepnovo_style_1	-use_spectrum_charge

Ursgal value translations

Ursgal Value	Translated Value		
.	msfragger_style_1	msfragger_style_2	msfragger_style_3
0	1	1	1
1	0	0	0

5.1.341 use_spectrum_mz

Does not correct precusor m/z.

Default value True

type bool

triggers rerun True

Available in unodes

- moda_v1_51
- moda_v1_61
- moda_v1_62
- pepnovo_3_1

Ursgal key translations for *use_spectrum_mz*

Style	Translation
moda_style_1	AutoPMCorrection
pepnovo_style_1	-use_spectrum_mz

Ursgal value translations

Ursgal Value	Translated Value
.	moda_style_1
0	1
1	0

5.1.342 validated_ident_csv_suffix

CSV suffix of validated identification files: string, CSV-file which contains PSMs validated with validation tools

Default value validated.csv

type str

triggers rerun True

Available in unodes

- ucontroller

Ursgal key translations for *validated_ident_csv_suffix*

Style	Translation
ucontroller_style_1	validated_ident_csv_suffix

5.1.343 validation_generalized

Generalized target decoy competition, situations where PSMs known to more frequently be incorrect are mixed in with the correct PSMs

Default value False

type bool

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *validation_generalized*

Style	Translation
qvality_style_1	-g

Ursgal value translations

Ursgal Value	Translated Value
.	qvality_style_1
0	None
1	

5.1.344 validation_minimum_score

Defines the minimum score used for validation. If scores lower than this are produced, they are set to the minimum score. This i

'None' : None

Default value None

type str

triggers rerun True

Available in unodes

- qvality_2_02

Ursgal key translations for *validation_minimum_score*

Style	Translation
qvality_style_1	validation_minimum_score

Ursgal value translations

Ursgal Value	Translated Value
.	qvality_style_1
moda_v1_51	0
moda_v1_61	0
moda_v1_62	0
msamanda_1_0_0_5242	0
msamanda_1_0_0_5243	0
msamanda_1_0_0_6299	0
msamanda_1_0_0_6300	0
msamanda_1_0_0_7503	0
msamanda_1_0_0_7504	0
msamanda_2_0_0_10695	0
msamanda_2_0_0_11219	0
msamanda_2_0_0_13723	0
msamanda_2_0_0_14665	0
msamanda_2_0_0_9695	0
msamanda_2_0_0_9706	0
msfragger_20170103	0
msfragger_20171106	0
msfragger_20190222	0
msfragger_20190628	0
msfragger_2_3	0
msfragger_3_0	0
msgfplus_v2016_09_16	1e-100
msgfplus_v2017_01_27	1e-100
msgfplus_v2018_01_30	1e-100
msgfplus_v2018_06_28	1e-100
msgfplus_v2018_09_12	1e-100
msgfplus_v2019_01_22	1e-100
msgfplus_v2019_04_18	1e-100
msgfplus_v2019_07_03	1e-100
msgfplus_v9979	1e-100
myrimatch_2_1_138	0
myrimatch_2_2_140	0
omssa_2_1_9	1e-30
pipi_1_4_5	0
pipi_1_4_6	0
xtandem_alanine	0
xtandem_cyclone_2010	0
xtandem_jackhammer	0
xtandem_piledriver	0
xtandem_sledgehammer	0
xtandem_vengeance	0

5.1.345 validation_score_field

Name of the column that is used for validation, e.g. by qvality and percolator. If None is defined, default values are used
'None' : None

Default value None

type str

triggers rerun True

Available in unodes

- add_estimated_fdr_1_0_0
- percolator_2_08
- percolator_3_2_1
- percolator_3_4_0
- qvality_2_02
- sanitize_csv_1_0_0
- svm_1_0_0
- ucontroller
- unify_csv_1_0_0
- ptminer_1_0

Ursgal key translations for *validation_score_field*

Style	Translation
add_estimated_fdr_style_1	validation_score_field
percolator_style_1	validation_score_field
ptminer_style_1	validation_score_field
qvality_style_1	validation_score_field
sanitize_csv_style_1	validation_score_field
svm_style_1	validation_score_field
ucontroller_style_1	validation_score_field
unify_csv_style_1	validation_score_field

Ursgal value translations

Ursgal Value	Translated Value			
.	add_estimated_fdr_style_1	percolator_style_1	qvality_style_1	sanitize_csv_sty
deepnovo_0_0_1	DeepNovo:score	DeepNovo:score	DeepNovo:score	DeepNovo:score
deepnovo_pointnovo	DeepNovo:score	DeepNovo:score	DeepNovo:score	DeepNovo:score
mascot_x_x_x	Mascot:Score	Mascot:Score	Mascot:Score	Mascot:Score
moda_v1_51	ModA:probability	ModA:probability	ModA:probability	ModA:probabilit
moda_v1_61	ModA:probability	ModA:probability	ModA:probability	ModA:probabilit

Table 6 – continued from previous page

Ursgal Value	Translated Value			
.	add_estimated_fdr_style_1	percolator_style_1	qvality_style_1	sanitize_csv_sty
moda_v1_62	ModA:probability	ModA:probability	ModA:probability	ModA:probability
msamanda_1_0_0_5242	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_5243	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_6299	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_6300	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_7503	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_7504	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_10695	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_11219	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_13723	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_14665	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_9695	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_2_0_0_9706	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msfragger_20170103	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msfragger_20171106	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msfragger_20190222	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msfragger_20190628	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msfragger_2_3	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msfragger_3_0	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hype
msgfplus_v2016_09_16	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2017_01_27	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2018_01_30	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2018_06_28	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2018_09_12	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2019_01_22	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2019_04_18	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v2019_07_03	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
msgfplus_v9979	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEVa
myrimatch_2_1_138	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH
myrimatch_2_2_140	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH
novor_1_05	Novor:score	Novor:score	Novor:score	Novor:score
novor_1_1beta	Novor:score	Novor:score	Novor:score	Novor:score
omssa_2_1_9	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue
pepnovo_3_1	Pepnovo:PnvScr	Pepnovo:PnvScr	Pepnovo:PnvScr	Pepnovo:PnvScr
pglyco_db_2_2_0	pGlyco:TotalScore	pGlyco:TotalScore	pGlyco:TotalScore	pGlyco:TotalScor
pglyco_db_2_2_2	pGlyco:TotalScore	pGlyco:TotalScore	pGlyco:TotalScore	pGlyco:TotalScor
pipi_1_4_5	PIPI:score	PIPI:score	PIPI:score	PIPI:score
pipi_1_4_6	PIPI:score	PIPI:score	PIPI:score	PIPI:score
pnovo_3_1_3	pNovo:Score	pNovo:Score	pNovo:Score	pNovo:Score
tag_graph_1_8_0	TagGraph:: 1-log10 EM	TagGraph:: 1-log10 EM	TagGraph:: 1-log10 EM	TagGraph:: 1-log
unknown	n/t	n/t	n/t	n/t
xtandem_alanine	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper
xtandem_cyclone_2010	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper
xtandem_jackhammer	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper
xtandem_piledriver	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper
xtandem_sledgehammer	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper
xtandem_vengeance	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyper

5.1.346 visualization_color_positions

Specifies colors for the datasets that should be visualized. Given as a dict in which the key represents the position of the corresponding dataset in the list, e.g.: {"0" : "#e41a1c", "1" : "#377eb8"}

Default value []

type dict

triggers rerun True

Available in unodes

- venndiagram_1_1_0

Ursgal key translations for *visualization_color_positions*

Style	Translation
venndiagram_style_1	visualization_color_position

5.1.347 visualization_column_names

The specified csv column names are used for the visualization. E.g. for a Venn diagram the entries of these columns are used (merged) to determine overlapping results.

Default value ['Modifications', 'Sequence']

type list

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_column_names*

Style	Translation
venndiagram_style_1	visualization_column_names

5.1.348 visualization_font

Font used for visualization plots (e.g. Venn diagram), given as dict with keys: font_type, font_size_header, font_size_major, font_size_minor, font_size_venn

Default value ['font_size_header', 'font_size_major', 'font_size_minor', 'font_size_venn', 'font_type']

type dict

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_font*

Style	Translation
venndiagram_style_1	visualization_font

5.1.349 visualization_header

Header of visualization output (e.g. Venn diagram)

Default value

type str

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0
- sugarpy_plot_1_0_0

Ursgal key translations for *visualization_header*

Style	Translation
sugarpy_plot_style_1	title
venndiagram_style_1	header

5.1.350 visualization_label_positions

Specifies labels for the datasets that should be visualized. Given as a dict in which the key represents the position of the corresponding dataset in the list, e.g.: {“0” : “LabelA”, “1” : “LabelB”}

Default value []

type dict

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_label_positions*

Style	Translation
venndiagram_style_1	visualization_label_position

5.1.351 visualization_opacity

Opacity used in visualization plots (e.g. Venn diagram)

Default value 0.35

type float

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_opacity*

Style	Translation
venndiagram_style_1	opacity

5.1.352 visualization_scaling_factors

Scaling factor for visualization plots (e.g. Venn diagram), given as dict with keys: x_axis, y_axis

Default value ['x_axis', 'y_axis']

type dict

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_scaling_factors*

Style	Translation
venndiagram_style_1	visualization_scaling_factors

5.1.353 visualization_size

Size of visualization plots (e.g. Venn diagram), given as dict with keys: width, height

Default value ['height', 'width']

type dict

triggers rerun True

Available in unodes

- venndiagram_1_0_0
- venndiagram_1_1_0

Ursgal key translations for *visualization_size*

Style	Translation
venndiagram_style_1	visualization_size

5.1.354 visualization_stroke_width

Stroke width used in visualization plots (e.g. Venn diagram)

Default value 2.0

type float

triggers rerun True

CHAPTER 6

Engines

Ursgal allows existing programs to be incorporated with ease. We call those programs or scripts engines. Currently, ursgal comes with these engines:

6.1 Included engines

6.1.1 Available Search Engines

Protein Database Search Engines

MS Amanda

Available MS Amanda versions, starting with the newest version:

class ursgal.wrappers.msamanda_2_0_0_14665.**msamanda_2_0_0_14665**(**args*, ***kwargs*)

MSAmanda 2_0_0_14665 UNode

Import functions from msamanda_2_0_0_9695

This third party engine can be downloaded from: https://ms.imp.ac.at/?goto=msamanda

class ursgal.wrappers.msamanda_2_0_0_13723.**msamanda_2_0_0_13723**(**args*, ***kwargs*)

MSAmanda 2_0_0_13723 UNode

Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_11219.**msamanda_2_0_0_11219**(**args*, ***kwargs*)

MSAmanda 2_0_0_11219 UNode

Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_10695.**msamanda_2_0_0_10695**(**args*, ***kwargs*)

MSAmanda 2_0_0_9706 UNode

Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_9706.**msamanda_2_0_0_9706**(**args*, ***kwargs*)

MSAmanda 2_0_0_9706 UNode

Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_9695.**msamanda_2_0_0_9695**(**args*, ***kwargs*)

MSAmanda 2_0_0_9695 UNode Parameter options at http://ms.imp.ac.at/inc/pd-nodes/msamanda/Manual%20MS%20Amanda%20Standalone.pdf

Note: Please download and install MSAmanda manually from http://ms.imp.ac.at/?goto=msamanda

Reference: Dorfer V, Pichler P, Stranzl T, Stadlmann J, Taus T, Winkler S, Mechtler K. (2014) MS Amanda, a universal identification algorithm optimized for high accuracy tandem mass spectra.

postflight()

Convert .tsv result files to .csv

preflight()

Formatting the command line via self.params

Settings file is created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params(dict)

class ursgal.wrappers.msamanda_1_0_0_7504.**msamanda_1_0_0_7504**(**args*, ***kwargs*)

MSAmanda 1_0_0_7504 UNode

Import functions from msamanda_1_0_0_5243

class ursgal.wrappers.msamanda_1_0_0_7503.**msamanda_1_0_0_7503**(**args*, ***kwargs*)

MSAmanda 1_0_0_7503 UNode

Import functions from msamanda_1_0_0_5243

class ursgal.wrappers.msamanda_1_0_0_5243.**msamanda_1_0_0_5243**(**args*, ***kwargs*)

MSAmanda 1_0_0_5243 UNode Parameter options at http://ms.imp.ac.at/inc/pd-nodes/msamanda/Manual%20MS%20Amanda%20Standalone.pdf

Reference: Dorfer V, Pichler P, Stranzl T, Stadlmann J, Taus T, Winkler S, Mechtler K. (2014) MS Amanda, a universal identification algorithm optimized for high accuracy tandem mass spectra.

postflight()

Convert .tsv result files to .csv

preflight()

Formatting the command line via self.params

Settings file is created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params(dict)

class ursgal.wrappers.msamanda_1_0_0_5242.**msamanda_1_0_0_5242**(**args*, ***kwargs*)

MSAmanda 1_0_0_5242 UNode

Import functions from msamanda_1_0_0_5243

MS-GF+

Available MS-GF+ versions, starting with the newest version:

class ursgal.wrappers.msgfplus_v2019_07_03.**msgfplus_v2019_07_03**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

postflight()

This can be/is overwritten by the engine uNode class

preflight()

Formatting the command line via self.params

Modifications file will be created in the output folder

Returns self.params

Return type dict

class ursgal.wrappers.msgfplus_v2019_04_18.**msgfplus_v2019_04_18**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2019_01_22.**msgfplus_v2019_01_22**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2018_09_12.**msgfplus_v2018_09_12**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2018_06_28.**msgfplus_v2018_06_28**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2018_01_30.**msgfplus_v2018_01_30**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2017_01_27.**msgfplus_v2017_01_27**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2016_09_16.**msgfplus_v2016_09_16**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

postflight()

This can be/is overwritten by the engine uNode class

preflight()

Formatting the command line via self.params

Modifications file will be created in the output folder

Returns self.params

Return type dict

class ursgal.wrappers.msgfplus_v9979.**msgfplus_v9979**(**args*, ***kwargs*)

MSGF+ UNode Parameter options at https://bix-lab.ucsd.edu/pages/viewpage.action?pageId=13533355

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

preflight()

Formatting the command line via self.params

Modifications file will be created in the output folder

Returns self.params

Return type dict

MSFragger

Available MSFragger versions, starting with the newest version:

class ursgal.wrappers.msfragger_3_0.**msfragger_3_0**(**args*, ***kwargs*)

MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

postflight()
Reads MSFragger tsv output and write final csv output file.

Adds:

- Raw data location, since this can not be added later
- Converts masses in Da to m/z (could be done in unify_csv)

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

class ursgal.wrappers.msfragger_2_3.**msfragger_2_3**(**args*, ***kwargs*)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

class ursgal.wrappers.msfragger_20190628.**msfragger_20190628**(**args*, ***kwargs*)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

postflight()
Reads MSFragger tsv output and write final csv output file.

Adds:

- Raw data location, since this can not be added later
- Converts masses in Da to m/z (could be done in unify_csv)

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

class ursgal.wrappers.msfragger_20190222.**msfragger_20190222**(**args*, ***kwargs*)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

class ursgal.wrappers.msfragger_20171106.**msfragger_20171106**(**args*, ***kwargs*)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

class ursgal.wrappers.msfragger_20170103.**msfragger_20170103**(**args*, ***kwargs*)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Kong, A. T., Leprevost, F. V, Avtonomov, D. M., Mellacheruvu, D., and Nesvizhskii, A. I. (2017) MSFragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. Nature Methods 14

Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

postflight()
Reads MSFragger tsv output and write final csv output file.

Adds:

- Raw data location, since this can not be added later
- Converts masses in Da to m/z (could be done in unify_csv)

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

MODa

Available MODa versions, starting with the newest version:

class ursgal.wrappers.moda_v1_62.**moda_v1_62**(**args*, ***kwargs*)
MODa UNode Check http://prix.hanyang.ac.kr/download/moda.jsp for download, new versions and contact information

Reference: Na S, Bandeira N, Paek E. (2012) Fast multi-blind modification search through tandem mass spectrometry.

postflight()
Rewrite ModA output .tsv into .csv so that it can be unified

preflight()
Formatting the command line via self.params

Returns self.params

Return type dict

class ursgal.wrappers.moda_v1_61.**moda_v1_61**(**args*, ***kwargs*)
MODa UNode Check http://prix.hanyang.ac.kr/download/moda.jsp for download, new versions and contact information

Reference: Na S, Bandeira N, Paek E. (2012) Fast multi-blind modification search through tandem mass spectrometry.

postflight()
Rewrite ModA output .tsv into .csv so that it can be unified

preflight()
Formatting the command line via self.params

Returns self.params

Return type dict

class ursgal.wrappers.moda_v1_51.**moda_v1_51**(**args*, ***kwargs*)
MODa UNode Check http://prix.hanyang.ac.kr/download/moda.jsp for download, new versions and contact information

Reference: Na S, Bandeira N, Paek E. (2012) Fast multi-blind modification search through tandem mass spectrometry.

postflight()
Rewrite ModA output .tsv into .csv so that it can be unified

preflight()
Formatting the command line via self.params

Returns self.params

Return type dict

MyriMatch

class ursgal.wrappers.myrimatch_2_1_138.**myrimatch_2_1_138**(**args*, ***kwargs*)
Myrimatch UNode

Myrimatch options: http://forge.fenchurch.mc.vanderbilt.edu/scm/viewvc.php/*checkout*/trunk/doc/index.html?root=myrimatch

Reference: Tabb DL, Fernando CG, Chambers MC. (2007) MyriMatch: highly accurate tandem mass spectral peptide identification by multivariate hypergeometric analysis.

postflight()
renaming MyriMatch's output file to our desired output file name

preflight()
Formatting the command line

write_param_file()
Writes a file containing all parameters for the search

class ursgal.wrappers.myrimatch_2_2_140.**myrimatch_2_2_140**(**args*, ***kwargs*)
Myrimatch UNode

Import functions from myrimatch_2_1_138

TagGraph

This open modification search engine uses de novo search engine results but takes a protein database into account as well.

class ursgal.wrappers.tag_graph_1_8_0.**tag_graph_1_8_0**(**args*, ***kwargs*)
TagGraph unode For further information see https://sourceforge.net/projects/taggraph/

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html

Reference: Devabhaktuni, A.; Lin, S.; Zhang, L.; Swaminathan, K.; Gonzalez, CG.; Olsson, N.; Pearlman, SM.; Rawson, K.; Elias, JE. (2019) TagGraph reveals vast protein modification landscapes from large tandem mass spectrometry datasets. Nat Biotechnol. 37(4)

postflight()
Reads TagGraph tdv output and write final csv output file.

preflight()
Formatting the command line and writing two param input files via self.params

Returns self.params

Return type dict

OMSSA

class ursgal.wrappers.omssa_2_1_9.**omssa_2_1_9**(**args*, ***kwargs*)
omssa_2_1_9 UNode

Parameter options at http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/asn_spec/omssa.asn.html

OMSSA 2.1.9 parameters at http://proteomicsresource.washington.edu/protocols06/omssa.php

Reference: Geer LY, Markey SP, Kowalak JA, Wagner L, Xu M, Maynard DM, Yang X, Shi W, Bryant SH (2004) Open Mass Spectrometry Search Algorithm.

postflight()
Will correct the OMSSA headers and add the column retention time to the csv file

preflight()
Formatting the command line via self.params

unimod Modifications are translated to OMSSA modifications

Returns self.params(dict)

PIPI

Available PIPI versions, starting with the newest version:

class ursgal.wrappers.pipi_1_4_6.**pipi_1_4_6**(**args*, ***kwargs*)
Unode for PIPI: PTM-Invariant Peptide Identification For furhter information see: http://bioinformatics.ust.hk/pipi.html

Note: Please download and extract PIPI manually from http://bioinformatics.ust.hk/pipi.html

Reference: Yu, F., Li, N., Yu, W. (2016) PIPI: PTM-Invariant Peptide Identification Using Coding Method. J Prot Res 15(12)

class ursgal.wrappers.pipi_1_4_5.**pipi_1_4_5**(**args*, ***kwargs*)
Unode for PIPI: PTM-Invariant Peptide Identification For furhter information see: http://bioinformatics.ust.hk/pipi.html

Note: Please download and extract PIPI manually from http://bioinformatics.ust.hk/pipi.html

Reference: Yu, F., Li, N., Yu, W. (2016) PIPI: PTM-Invariant Peptide Identification Using Coding Method. J Prot Res 15(12)

postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

X!Tandem

Available X!Tandem versions, starting with the newest version:

class ursgal.wrappers.xtandem_alanine.**xtandem_alanine**(**args*, ***kwargs*)
X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/

Reference: Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem mass spectra.

class ursgal.wrappers.xtandem_vengeance.**xtandem_vengeance**(**args*, ***kwargs*)
X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/

Reference: Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem mass spectra.

format_templates()
Returns formatted X!Tandem input files

The formating is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line via self.params

Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params

Return type dict

class ursgal.wrappers.xtandem_piledriver.**xtandem_piledriver**(**args*, ***kwargs*)

X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/

Reference: Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem mass spectra.

format_templates()

Returns formatted X!Tandem input files

The formating is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

postflight()

This can be/is overwritten by the engine uNode class

preflight()

Formatting the command line via self.params

Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params

Return type dict

class ursgal.wrappers.xtandem_sledgehammer.**xtandem_sledgehammer**(**args*, ***kwargs*)

X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/

Reference: Craig R, Beavis RC. (2004) TANDEM: matching proteins with tandem mass spectra.

format_templates()

Returns formatted X!Tandem input files

The formating is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

postflight()

This can be/is overwritten by the engine uNode class

preflight()

Formatting the command line via self.params

Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params

Return type dict

class ursgal.wrappers.xtandem_jackhammer.**xtandem_jackhammer**(**args*, ***kwargs*)

class ursgal.wrappers.xtandem_cyclone_2010.**xtandem_cyclone_2010**(**args*, ***kwargs*)

De Novo Search Engines

DeepNovo

Available DeepNovo versions, starting with the newest version:

class ursgal.wrappers.deepnovo_pointnovo.**deepnovo_pointnovo**(**args*, ***kwargs*)

PointNovo UNode pytorch re-implementation of DeepNovo For further information, see https://github.com/volpato30/PointNovo/

Reference: Tran, N.H.; Zhang, X.; Xin, L.; Shan, B.; Li, M. (2017) De novo peptide sequencing by deep learning. PNAS 114 (31)

postflight()

Reformats the DeepNovo output file

preflight()

Create deepnovo_config.py file and format command line via self.params

Returns self.params

Return type dict

class ursgal.wrappers.deepnovo_0_0_1.**deepnovo_0_0_1**(**args*, ***kwargs*)

DeepNovo UNode For further information, see https://github.com/nh2tran/DeepNovo

---

Note: Please download manually from https://github.com/StSchulze/DeepNovo?organization=StSchulze&organization=StSchulze or using git clone https://github.com/StSchulze/DeepNovo.git and download the model from https://drive.google.com/open?id=0By9IxqHK5MdWalJLSGliWW1RY2c

---

Reference: Tran, N.H.; Zhang, X.; Xin, L.; Shan, B.; Li, M. (2017) De novo peptide sequencing by deep learning. PNAS 114 (31)

postflight()

Reformats the DeepNovo output file

preflight()

Create deepnovo_config.py file and format command line via self.params

Returns self.params

Return type dict

Novor

Available Novor versions, starting with the newest version:

class ursgal.wrappers.novor_1_05.**novor_1_05**(**args*, ***kwargs*)

Novor UNode Parameter options at http://rapidnovor.com/

Reference: Bin Ma (2015) Novor: Real-Time Peptide de Novo Sequencing Software. J Am Soc Mass Spectrom 26 (11)

Import node for version novor_1_1beta

postflight()

Reformats the Novor output file

preflight()
Formatting the command line via self.params

Params.txt file will be created in the output folder

Returns self.params

Return type dict

class ursgal.wrappers.novor_1_1beta.**novor_1_1beta**(**args*, ***kwargs*)
Novor UNode Parameter options at http://rapidnovor.com/

Reference: Bin Ma (2015) Novor: Real-Time Peptide de Novo Sequencing Software.

postflight()
Reformats the Novor output file

preflight()
Formatting the command line via self.params

Params.txt file will be created in the output folder

Returns self.params

Return type dict

PepNovo

class ursgal.wrappers.pepnovo_3_1.**pepnovo_3_1**(**args*, ***kwargs*)
PepNovo v3.1 UNode http://proteomics.ucsd.edu/Software/PepNovo/

Reference: Ari M. Frank, Mikhail M. Savitski, Michael L. Nielsen, Roman A. Zubarev, and Pavel A. Pevzner (2007) De Novo Peptide Sequencing and Identification with Precision Mass Spectrometry, J. Proteome Res. 6:114-123.

postflight()
Reformats the PepNovo output file

preflight()
Formatting the command line via self.params

Returns self.params

Return type dict

pNovo

class ursgal.wrappers.pnovo_3_1_3.**pnovo_3_1_3**(**args*, ***kwargs*)
Unode for pNovo 3.1.3 For furhter information see: http://pfind.ict.ac.cn/software/pNovo/

Note: Please download pNovo 3.1.3 manually from http://pfind.ict.ac.cn/software/pNovo/#Downloads

Reference: Yang, H; Chi, H; Zhou, W; Zeng, WF; He, K; Liu, C; Sun, RX; He, SM. (2017) Open-pNovo: De Novo Peptide Sequencing with Thousands of Protein Modifications. J Proteome Res. 16(2)

postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

Cross Link Search Engines

Kojak

class ursgal.wrappers.kojak_1_5_3.**kojak_1_5_3**(**args*, ***kwargs*)
Kojak UNode Parameter options at http://www.kojak-ms.org/param/index.html

Reference: Hoopmann MR, Zelter A, Johnson RS, Riffle M, Maccoss MJ, Davis TN, Moritz RL (2015) Kojak: Efficient analysis of chemically cross-linked protein complexes. J Proteome Res 14: 2190-198

Note: Kojak has to be installed manually at the moment! Use folder name: 'kojak_1_5_3' in the resources folder.

format_templates()
Returns formatted input files as a dict.

The standard parametern file is used and adjustes.

Returns keys are the names of the parametern template file

Return type dict

postflight()
Move the result files to the Kojak folder, since the output files can not be specified manually.

preflight()
Formatting the command line via self.params

Glycosylation Search Engines

pGlyco

Available pGlyci versions, starting with the newest version:

class ursgal.wrappers.pglyco_db_2_2_2.**pglyco_db_2_2_2**(**args*, ***kwargs*)
Unode for pGlyco 2.2.2 For furhter information see: https://github.com/pFindStudio/pGlyco2

Note: Please download pGlyco 2.2.2 manually from https://github.com/pFindStudio/pGlyco2

Reference: Liu MQ, Zeng WF,, Fang P, Cao WQ, Liu C, Yan GQ, Zhang Y, Peng C, Wu JQ, Zhang XJ, Tu HJ, Chi H, Sun RX, Cao Y, Dong MQ, Jiang BY, Huang JM, Shen HL, Wong CCL, He SM, Yang PY. (2017) pGlyco 2.0 enables precision N-glycoproteomics with comprehensive quality control and one-step mass spectrometry for intact glycopeptide identification. Nat Commun 8(1)

class ursgal.wrappers.pglyco_db_2_2_0.**pglyco_db_2_2_0**(**args*, ***kwargs*)
Unode for pGlyco 2.2.0 For furhter information see: https://github.com/pFindStudio/pGlyco2

Note: Please download pGlyco 2.2.0 manually from https://github.com/pFindStudio/pGlyco2

Reference: Liu MQ, Zeng WF,, Fang P, Cao WQ, Liu C, Yan GQ, Zhang Y, Peng C, Wu JQ, Zhang XJ, Tu HJ, Chi H, Sun RX, Cao Y, Dong MQ, Jiang BY, Huang JM, Shen HL, Wong CCL, He SM, Yang PY. (2017) pGlyco 2.0 enables precision N-glycoproteomics with comprehensive quality control and one-step mass spectrometry for intact glycopeptide identification. Nat Commun 8(1)

postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

6.1.2 Converter Engines

Convert CSV to SSL 1_0_0

class ursgal.wrappers.csv2ssl_1_0_0.**csv2ssl_1_0_0**(**args*, ***kwargs*)
csv2ssl_1_0_0 UNode

_execute()
Result files (.csv) are converted to spectrum sequence list (.ssl) files. These .ssl can be used as input files for BiblioSpec.

Input file has to be a .csv

Creates a _converted.csv file and returns its path.

ursgal.resources.platform_independent.arc_independent.csv2ssl_1_0_0.csv2ssl_1_0_0.**main**(*input*
out-
put_f
score
score

Convert csvs to ssl

Convert CSV to Counted Results

class ursgal.wrappers.csv2counted_results_1_0_0.**csv2counted_results_1_0_0**(**args*, ***kwargs*)
csv2counted_results_1_0_0 UNode

_execute()
Results (.csv) are summarized as table (.csv) containing all identified proteins, peptides, or other specified identifiers. For each sample, the peptide or spectral count for each identifier is given.

Input file has to be a .csv

Creates a _counted.csv file and returns its path.

Columns containing the elements that should be counted (identifiers) are given as a list of headers using uc.params["identifier_column_names"]. Columns defining a unique countable element (e.g. "Sequence", "Spectrum ID") are given as a list of headers using uc.params["count_column_names"].

This can be used to create a SFINX (http://sfinx.ugent.be/) input file, using:

uc.params["convert_to_sfinx"]=True uc.params["identifier_colum_names"]=["Protein ID"]
uc.params["count_column_names"]=["Sequence"]

ursgal.resources.platform_independent.arc_independent.csv2counted_results_1_0_0.csv2counte

Results (.csv) are summarized as table (.csv) containing all identified proteins, peptides, or other specified identifiers. For each sample, the peptide or spectral count for each identifier is given.

This can be used to convert .csv files to SFINX input files.

This is a .csv file containing unique peptide counts for all identified proteins. However, this can be modified using the keywords "identifier_colum_names" and "count_column_names"

Keyword Arguments

- **input_file** (*str*) – name including path for the input file
- **output_file** (*str*) – name including path for the output file
- **identifier_colum_names** (*list*) – list of column headers that define the identifier. Multiple column names are joined for combined identifiers.
- **count_column_names** (*list*) – list of column headers which are used for counting.
- **count_by_file** (*bool*) – the number of unique hits for each identifier is given in seperate columns for each raw file (file name as defiened in Spectrum Title)
- **convert2sfinx** (*bool*) – If True, the header of the identifier column is "rownames". If False, the joined header name will be used.
- **keep_column_names** (*list*) – list of column headers which are not used as identifiers but kept in the output, e.g. when counting ['Sequence', 'Modifications'] the column ['Protein ID'] could be specified here. Multiple entries for one identifier (e.g. when identifier_column_names = ['Potein ID'] and keep_column_names = ['Sequence']) are seperated by '<#>'.

Convert Mascot DAT to CSV

class ursgal.wrappers.mascot_dat2csv_1_0_0.**mascot_dat2csv_1_0_0**(**args*, ***kwargs*)

Dummy to merge mascot data into usgal workflow

ursgal.resources.platform_independent.arc_independent.mascot_dat2csv_1_0_0.mascot_dat2csv_1

Convert MS-GF+ MZID to CSV

class ursgal.wrappers.msgfplus2csv_py_v1_0_0.**msgfplus2csv_py_v1_0_0**(**args*, ***kwargs*)

msgfplus2csv_py v1.0.0 UNode

class ursgal.wrappers.msgfplus2csv_v1_2_1.**msgfplus2csv_v1_2_1**(**args*, ***kwargs*)

msgfplus2csv_v1.2.1 UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

postflight()

Convert .tsv result file to .csv and translates headers

preflight()

mzid result files from MS-GF+ are converted to CSV using the MzIDToTsv converter from MS-GF+

Input file has to be a .mzid or .mzid.gz

Creates a .csv file and returns its path

Mzid to Tsv Converter Usage: MzidToTsvConverter -mzid:"mzid path" [-tsv:"tsv output path"] [-unroll|-u] [-showDecoy|-sd]

Required parameters: '-mzid:path' - path to mzid[.gz] file; if path has spaces, it must be in quotes.

Optional parameters: '-tsv:path' - path to tsv file to be written; if not specified, will be output to same location as mzid '-unroll|-u' signifies that results should be unrolled - one line per unique peptide/protein combination in each spectrum identification '-showDecoy|-sd' signifies that decoy results should be included in the result tsv

class ursgal.wrappers.msgfplus2csv_v1_2_0.**msgfplus2csv_v1_2_0**(**args*, ***kwargs*)

msgfplus_C_mzid2csv_v1.2.0 UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

class ursgal.wrappers.msgfplus2csv_v2017_07_04.**msgfplus2csv_v2017_07_04**(**args*, ***kwargs*)

msgfplus_C_mzid2csv_v2017_07_04 UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

postflight()

Convert .tsv result file to .csv and translates headers

preflight()

mzid result files from MS-GF+ are converted to CSV using the MzIDToTsv converter from MS-GF+

Input file has to be a .mzid or .mzid.gz

Creates a .csv file and returns its path

Mzid to Tsv Converter Usage: MzidToTsvConverter -mzid:"mzid path" [-tsv:"tsv output path"] [-unroll|-u] [-showDecoy|-sd]

Required parameters: '-mzid:path' - path to mzid[.gz] file; if path has spaces, it must be in quotes.

Optional parameters: '-tsv:path' - path to tsv file to be written; if not specified, will be output to same location as mzid '-unrolll-u' signifies that results should be unrolled - one line per unique peptide/protein combination in each spectrum identification '-showDecoyl-sd' signifies that decoy results should be included in the result tsv

class ursgal.wrappers.msgfplus2csv_v2017_01_27.**msgfplus2csv_v2017_01_27**(**args*, ***kwargs*)

msgfplus2csv_v2017_01_27 UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference:

Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

class ursgal.wrappers.msgfplus2csv_v2016_09_16.**msgfplus2csv_v2016_09_16**(**args*, ***kwargs*)

msgfplus2csv_v2016_09_16 UNode Parameter options at https://omics.pnl.gov/software/ms-gf

Reference: Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: Applications to Database Search.

postflight()

Convert .tsv result file to .csv

preflight()

mzid result files from MS-GF+ are converted to CSV using the MzIDToTsv converter from MS-GF+

Input file has to be a .mzid

Creates a .csv file and returns its path

Convert MZML to MGF

The mzML to mgf converter version 2.0.0 requires pymzML 2.0, while the previous version can be used with older pymzML versions.

class ursgal.wrappers.mzml2mgf_2_0_0.**mzml2mgf_2_0_0**(**args*, ***kwargs*)

mzml2mgf_2_0_0 UNode

Version two works only with pymzML version 2.0.0 or higher!

Converts .mzML files into .mgf files

ursgal.resources.platform_independent.arc_independent.mzml2mgf_2_0_0.mzml2mgf_2_0_0.**main**(*m*
m
*i_*
m
m
ch
sc
sc
pr
fix
sc
m
pr
cu
so
pr
cu
so
io
sp
si
na

class ursgal.wrappers.mzml2mgf_1_0_0.**mzml2mgf_1_0_0**(**args*, ***kwargs*)
mzml2mgf_1_0_0 UNode

Converts .mzML files into .mgf files

Convert X!Tandem XML to CSV 1_0_0

class ursgal.wrappers.xtandem2csv_1_0_0.**xtandem2csv_1_0_0**(**args*, ***kwargs*)
xtandem2csv_1_0_0 UNode

ursgal.resources.platform_independent.arc_independent.xtandem2csv_1_0_0.xtandem2csv_1_0_0.**r**

Converts xTandem.xml files into .csv We need to do this on our own, because mzidentml_lib reports wrong positions for modifications (and it is also not able to convert the piledriver.mzid into csv)

It should be noted that - xtandem groups are not merged (since it is not the same as protein groups) - multiple domains (multiple occurence of a peptide in the same protein) are not reported

MzidLib

class ursgal.wrappers.mzidentml_lib_1_7.**mzidentml_lib_1_7**(**args*, ***kwargs*)
MzidLib 1_7UNode

Import functions from mzidentml_lib_1_6_10

Note: Please download and install manually from http://www.proteoannotator.org/?q=installation

class ursgal.wrappers.mzidentml_lib_1_6_11.**mzidentml_lib_1_6_11**(**args*, ***kwargs*)

MzidLib 1_6_11 UNode

Import functions from mzidentml_lib_1_6_10

class ursgal.wrappers.mzidentml_lib_1_6_10.**mzidentml_lib_1_6_10**(**args*, ***kwargs*)

MzidLib 1_6_10 UNode

'Reisinger F, Krishna R, Ghali F, Ríos D, Hermjakob H, Vizcaíno JA, Jones AR. (2012) jmzIdentML API: A Java interface to the mzIdentML standard for peptide and protein identification data.'

Java program to convert results to .mzIdentML and .mzIdentML to .csv

preflight()

Convert .mzid result files from different search engines into .csv result files

For X!Tandem result files first need to be converted into .mzid with raw2mzid

raw2mzid(*search_engine=None*, *translations=None*)

Convert raw result files into .mzid result files

pParse 2.0

class ursgal.wrappers.pparse_2_0.**pparse_2_0**(**args*, ***kwargs*)

Unode for pParse included in pGlyco 2.2.0 For further information visit http://pfind.ict.ac.cn/software/pParse/#Downloads

Note: Please download pParse manually as part of pGlyco 2.2.0 https://github.com/pFindStudio/pGlyco2

Reference: Yuan ZF, Liu C, Wang HP, Sun RX, Fu Y, Zhang JF, Wang LH, Chi H, Li Y, Xiu LY, Wang WP, He SM (2012) pParse: a method for accurate determination of monoisotopic peaks in high-resolution mass spectra. Proteomics 12(2)

postflight()

Rename output file, since that naming the output file is not properly working in pParse

preflight()

Formatting the command line via self.params

Returns self.params

Return type dict

Command line options:

-D datapath	default (D:data)
-L logfilepath	default (the same with datapath)
-O outputpath	default (the same with datapath)
-W isolation_width	default (2)
-F input_format	default (raw) optional (wiff)
-C co-elute	default (1)
-S cut_similiar_mono	default (1)
-I ipv_file	default (.IPV.txt)

-M mars_model default (4)

-T trainingset default (.TrainingSet.txt)

-m output_mgf default (1)

-p output_pf default (1)

-d delete_msn default (0)

-a check_activationcenter default (1) -g debug_mode default (0) -r rewrite_files default (0) -t mars_threshold default (-0.34) -u export_unchecked_mono default (0) -y output_all_mars_y default (0) -Y output_mars_y default (0) -s output_trainingdata default (0) -R recalibrate_window default (7) -v outputsvmlight default (0) -z m/z default (5) -i Intensity default (1)

ThermoRawFileParser

class ursgal.wrappers.thermo_raw_file_parser_1_1_2.**thermo_raw_file_parser_1_1_2**(**args*, ***kwargs*)

Unode for ThermoRawFileParser For further information visit https://github.com/compomics/ThermoRawFileParser

---

Note: Please download ThermoRawFileParser manually from https://github.com/compomics/ThermoRawFileParser

---

Reference: Hulstaert N, Sachsenberg T, Walzer M, Barsnes H, Martens L and Perez-Riverol Y (2019) ThermoRawFileParser: modular, scalable and cross-platform RAW file conversion. bioRxiv https://doi.org/10.1101/622852

preflight()

Formatting the command line via self.params

Returns self.params

Return type dict

ThermoRawFileParser.exe usage is (use -option=value for the optional arguments):

-h, --help Prints out the options.

-i, --input=VALUE The raw file input.

-o, --output=VALUE The output directory.

-f, --format=VALUE The output format for the spectra (0 for MGF, 1 for mzML, 2 for indexed mzML, 3 for Parquet, 4 for MGF with profile data excluded)

-m, --metadata=VALUE The metadata output format (0 for JSON, 1 for TXT).

-g, --gzip GZip the output file if this flag is specified (without value).

-u, –s3_url[=VALUE] Optional property to write directly the data into S3 Storage.

-k, –s3_accesskeyid[=VALUE]

Optional key for the S3 bucket to write the file output.

-t, –s3_secretaccesskey[=VALUE]

Optional key for the S3 bucket to write the file output.

-n, –s3_bucketName[=VALUE] S3 bucket name

-v, --verbose Enable verbose logging.

-e, --ignoreInstrumentErrors Ignore missing properties by the instrument.

6.1.3 Other Engines

Fetcher

Get FTP Files 1_0_0

class ursgal.wrappers.get_ftp_files_1_0_0.**get_ftp_files_1_0_0**(**args*, ***kwargs*)

get_ftp_files_1_0_0 UNode

Downloads files from FTP servers

Note: meta info param 'output_extensions' is by default txt, so that the temporary txt json files get properly deleted

---

Parameters

- **ftp_url** (*) –
- **folder** (*) –
- **login** (*) –
- **password** (*) –
- **include_ext** (*) –
- **output_folder** (*) –
- **max_number_of_files** (*) –
- **blocksize** (*) –

_execute()

Downloads files from FTP server

ursgal.resources.platform_independent.arc_independent.get_ftp_files_1_0_0.get_ftp_files_1_0

Get HTTP Files 1_0_0

class ursgal.wrappers.get_http_files_1_0_0.**get_http_files_1_0_0**(**args*, ***kwargs*)

get_http_files_1_0_0 UNode

Downloads files via http

Parameters

- **http_url** (*) –
- **http_output_folder** (*) –

Note: meta info param 'output_extensions' is by default txt, so that the temporary txt json files get properly deleted

---

_execute()

Downloads files via http

ursgal.resources.platform_independent.arc_independent.get_http_files_1_0_0.get_http_files_1

Meta Engines

Combine FDR 0_1

class ursgal.wrappers.combine_FDR_0_1.**combine_FDR_0_1**(**args*, ***kwargs*)

combine FDR 0_1 UNode

An implementation of the "combined FDR Score" algorithm, as described in: Jones AR, Siepen JA, Hubbard SJ, Paton NW (2009): "Improving sensitivity in proteome studies by analysis of false discovery rates for multiple search engines."

Input should be multiple CSV files from different search engines. Each CSV requires a PEP column, for instance by post-processing with Percolator.

Returns a merged CSV file with all PSMs that were found and an added column "Combined FDR Score".

_execute()

Executing the combine_FDR_0_1 main function with parameters that were defined in preflight (stored in self.command_dict)

The main function is imported and then executed using the parameters from command_dict.

Returns None

preflight()

Building the list of parameters that will be passed to the combine_FDR_0_1 main function.

These parameters are stored in self.command_dict

Returns None

Combine PEP 1_0_0

class ursgal.wrappers.combine_pep_1_0_0.**combine_pep_1_0_0**(**args*, ***kwargs*)

combine_pep_1_0_0 UNode

Combining Multiengine Search Results with "Combined PEP"

"Combined PEP" is a hybrid approach combining elements of the "combined FDR" approach (Jones et al., 2009), elements of PeptideShaker, and elements of Bayes' theorem. Similar to "combined FDR", "combined PEP" groups the PSMs. For each search engine, the reported PSMs are treated as a set and the logical combinations of all sets are treated separately as done in the "combined FDR" approach. For instance, three search engines would result in seven PSM groups, which can be visualized by the seven intersections of a three-set Venn diagram. Typically, a PSM group that is shared by multiple engines contains fewer decoy hits and thus represents a higher quality subset and thus its PSMs receive a higher score. This approach is based on the assumption that the search engines agree on the decoys and false-positives as they agree on the targets.

The combined PEP approach uses Bayes' theorem to calculate a multiengine PEP (MEP) for each PSM based on the PEPs reported by, for example, Percolator for different search engines, that is

This is done for each PSM group separately.

Then, the combined PEP (the final score) is computed similar to PeptideShaker using a sliding window over all PSMs within each group (sorted by MEP). Each PSM receives a PEP based on the target/decoy ratio of the surrounding PSMs.

Finally, all groups are merged and the results reported in one output, including all the search result scores from the individual search engines as well as the FDR based on the "combined PEP".

The sliding window size can be defined by adjusting the Ursgal parameter "window_size" (default is 249).

Input should be multiple CSV files from different search engines. Each CSV requires a PEP column, for instance by post-processing with Percolator.

Returns a merged CSV file with all PSMs that were found and two added columns:

- **column "Bayes PEP":** The multi-engine PEP, see explanation above
- **column "combined PEP":** The PEP as computed within the engine combination PSMs

For optimal ranking, PSMs should be sorted by combined PEP. Ties can be resolved by sorting them by Bayes PEP.

_execute()
Executing the combine_FDR_0_1 main function with parameters that were defined in preflight (stored in self.command_dict)

The main function is imported and then executed using the parameters from command_dict.

Returns None

preflight()
Building the list of parameters that will be passed to the combine_pep_1_0_0 main function.

These parameters are stored in self.command_dict

Returns None

Misc Engines

Filter CSV 1_0_0

class ursgal.wrappers.filter_csv_1_0_0.**filter_csv_1_0_0**(**args*, ***kwargs*)
filter_csv_1_0_0 UNode

Filters .csv files row-wise according to user-defined rules.

The filter rules have to be defined in the params. See the engine documentation for further information (*filter_csv_1_0_0._execute()*).

_execute()

Result files (.csv) are filtered for defined filter parameters.

Input file has to be a .csv

Creates a _accepted.csv file and returns its path. If defined also rejected entries are written to _rejected.csv.

Note: To write the rejected entries define 'write_unfiltered_results' as True in the parameters.

Available rules:

- lte
- gte
- lt
- gt
- contains
- contains_not
- equals
- equals_not
- regex

Example

```
>>> params = {
>>>     'csv_filter_rules':[
>>>         ['PEP', 'lte', 0.01],
>>>         ['Is decoy', 'equals', 'false']
>>>     ]
>>>}
```

The example above would filter for posterior error probabilities lower than or equal to 0.01 and filter out all decoy proteins.

Rules are defined as list of lists with the first list element as the column name/csv fieldname, the second list element the rule and the third list element the value which should be compared. Multiple rules can be applied, see example above. If the same fieldname should be filtered multiply (E.g. Sequence should not contain 'T' and 'Y'), the rules have to be defined separately.

Example

```
>>> params = {
>>>     'csv_filter_rules':[
>>>         ['Sequence','contains_not','T'],
>>>         ['Sequence','contains_not','Y']
>>>     ]
>>>}
```

lte:

'lower than or equal' (<=) value has to comparable i.e. float or int. Values are accepted if they are lower than or equal to the defined value. E.g. ['PEP','lte',0.01]

gte:

'greater than or equal' (>=) value has to comparable i.e. float or int. Values are accepted if they are greater than or equal to the defined value. E.g. ['Exp m/z','gte',180]

lt:

'lower than' (<=) value has to comparable i.e. float or int. Values are accepted if they are lower than the defined value. E.g. ['PEP','lt',0.01]

gt:

'greater than' (>=) value has to comparable i.e. float or int. Values are accepted if they are greater than the defined value. E.g. ['PEP','gt',0.01]

contains:

Substrings are checked if they are present in the the full string. E.g. ['Modifications','contains','Oxidation']

contains_not:

Substrings are checked if they are present in the the full string. E.g. ['Sequence','contains_not','M']

equals:

String comparison (==). Comparison has to be an exact match to pass. E.g. ['Is decoy','equals','false']. Floats and ints are not compared at the moment!

equals_not:

String comparison (!=). Comparisons differing will be rejected. E.g. ['Is decoy','equals_not','true']. Floats and ints are not compared at the moment!

regex:

Any regular expression matching is possible E.g. CT and CD motif search ['Sequence','regex','C[T|D]']

Note: Some spreadsheet tools interpret False and True and show them as upper case when opening the files, even if they are actually written in lower case. This is especially important for target and decoy filtering, i.e. ['Is decoy','equals','false']. 'false' has to be lower case, even if the spreadsheet tool displays it as 'FALSE'.

---

ursgal.resources.platform_independent.arc_independent.filter_csv_1_0_0.filter_csv_1_0_0.**ma**

Filters csvs

Generate Target Decoy 1_0_0

class ursgal.wrappers.generate_target_decoy_1_0_0.**generate_target_decoy_1_0_0**(**args*, ***kwargs*)

Generate Target Decoy 1_0_0 UNode

_execute()

Creates a target decoy database based on shuffling of peptides or complete reversing the protein sequence.

The engine currently available generates a very stringent target decoy database by peptide shuffling but also offers the possibility to simple reverse the protein sequence. The mode can be defined in the params with 'decoy_generation_mode'.

The shuffling peptide method is described below. As one of the first steps redundant sequences are filtered and the protein gets a tag which highlight its double occurence in the database. This ensures that no unequal distribution of target and decoy peptides is present. Further, every peptide is shuffled, while the amindo acids where the enzyme cleaves aremaintained at their original position. Every peptide is only shuffled once and the shuffling result is stored. As a result it is ensured that if a peptide occurs multiple times it is shuffled the same way. It is further ensured that unmutable peptides (e.g. 'RR' for trypsin) are not shuffled and are reported by the engine as unmutable peptides in a text file, so that they can be excluded in the further analysis. This way of generating a target decoy database lead to the fulfillment of the following quality criteria (Proteome Bioinformatics, Eds: S.J. Hubbard, A.R. Jones, Humana Press).

Quality criteria:

- every target peptide sequence has exactly one decoy peptide sequence
- equal amino acid distribution
- equal protein and peptide length
- equal number of proteins and peptides
- similar mass distribution
- no predicted peptides in common

Avaliable modes:

- **shuffle_peptide - stringent target decoy generation with shuffling** of peptides with maintaining the cleavage site amino acid.
- reverse_protein - reverses the protein sequence

Available enzymes and their cleavage site can be found in the knowledge base of generate_target_decoy_1_0_0.

ursgal.resources.platform_independent.arc_independent.generate_target_decoy_1_0_0.generate_

Merge CSVS 1_0_0

class ursgal.wrappers.merge_csvs_1_0_0.**merge_csvs_1_0_0**(**args*, ***kwargs*)
Merge CSVS 1_0_0 UNode

_execute()
Merges .csv files

for same header, new rows are appended

for different header, new columns are appended

ursgal.resources.platform_independent.arc_independent.merge_csvs_1_0_0.merge_csvs_1_0_0.**ma**

Merges ident csvs

Sanitize CSV 1_0_0

class ursgal.wrappers.sanitize_csv_1_0_0.**sanitize_csv_1_0_0**(**args*, ***kwargs*)
sanitize_csv_1_0_0 UNode

Result files (.csv) are sanitized following defined parameters. That means, for each spectrum PSMs are compared and the best spectrum (spectra) is (are) chosen.

The parameters have to be defined in the params. See the engine documentation for further information (*sanitize_csv_1_0_0._execute()*).

_execute()
Result files (.csv) are sanitized following defined parameters. That means, for each spectrum PSMs are compared and the best spectrum (spectra) is (are) chosen

Input file has to be a .csv

Creates a _sanitized.csv file and returns its path.

Note: If not specified, the validation_score_field and bigger_scores_better parameters are determined from the last engine. Therefore, if sanitize_csv_1_0_0 is applied to merged or processed result files, both parameters need to be specified.

Available parameters:

- **score_diff_threshold (float): minimum score difference between** the best PSM and the first rejected PSM of one spectrum
- **threshold_is_log10 (bool): True, if log10 scale has been used for** score_diff_threshold.
- **accept_conflicting_psms (bool): If True, multiple PSMs for one** spectrum can be reported if their score difference is below the threshold. If False, all PSMs for one spectrum are removed if the score difference between the best and secondbest PSM is not above the threshold, i.e. if there are conflicting PSMs with similar scores.
- **num_compared_psms (int): maximum number of PSMs (sorted by score,** starting with the best scoring PSM) that are compared
- **remove_redundant_psms (bool): If True, redundant PSMs (e.g.** the same identification reported by multiple engined) for the same spectrum are removed. An identification is defined by the combination of 'Sequence', 'Modifications' and 'Charge'.

ursgal.resources.platform_independent.arc_independent.sanitize_csv_1_0_0.sanitize_csv_1_0_0

Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score is accepted and only if the best hit has a PEP that is at least two orders of magnitude smaller than the others

SugarPy v1_0_0

Contains run and plot functionality

class ursgal.wrappers.sugarpy_run_1_0_0.**sugarpy_run_1_0_0**(**args*, ***kwargs*)
SugarPy - discovery-driven analysis of glycan compositions from IS-CID of intact glycopeptides.

_execute()
Run SugarPy on a given .mzML file based on identified peptides from an evidences.csv

Translated Ursgal parameters are passed to the SugarPy main function.

class ursgal.wrappers.sugarpy_plot_1_0_0.**sugarpy_plot_1_0_0**(**args*, ***kwargs*)
SugarPy - discovery-driven analysis of glycan compositions from IS-CID of intact glycopeptides.

_execute()
Run the SugarPy plotting functions on a given .mzML file based on identified glycopeptides from SugarPy result .csv or a defined plot_molecule_dict as well as the corresponding validated_results_list.pkl

Translated Ursgal parameters are passed to the SugarPy main function.

Unify CSV 1_0_0

class ursgal.wrappers.unify_csv_1_0_0.**unify_csv_1_0_0**(**args*, ***kwargs*)
unify_csv_1_0_0 UNode

Unifies the .csv files of converted search engine results. The corrections for each engine are listed in the node under ursgal/resources/platform_independent/arc_independent/unify_csv_1_0_0

_execute()
Result files from search engines are unified to contain the same informations in the same style

Input file has to be a .csv

Creates a _unified.csv file and returns its path

ursgal.resources.platform_independent.arc_independent.unify_csv_1_0_0.unify_csv_1_0_0.**main**

Parameters

- **input_file** (*str*) – input filename of csv which should be unified
- **output_file** (*str*) – output filename of csv after unifying
- **scan_rt_lookup** (*dict*) – dictionary with entries of scanID to retention time under key 'scan_2_rt'
- **force** (*bool*) – force True or False
- **params** (*dict*) – params as passed by ursgal
- **search_engine** (*str*) – the search engine the csv file stems from
- **score_colname** (*str*) – the column names of the search engine's score (i.e. 'OMSSA:pvalue')

List of fixes

All engines

- Retention Time (s) is correctly set using _ursgal_lookup.pkl During mzML conversion to mgf the retention time for every spec is stored in a internal lookup and used later for setting the RT.
- All modifications are checked if they were given in params['modifications'], converted to the name that was given there and sorted according to their position.
- Fixed modifications are added in 'Modifications', if not reported by the engine.
- The monoisotopic m/z for for each line is calculated (uCalc m/z), since not all engines report the monoisotopic m/z
- Mass accuracy calculation (in ppm), also taking into account that not always the monoisotopic peak is picked
- Rows describing the same PSM (i.e. when two proteins share the same peptide) are merged to one row.

X!Tandem

- 'RTINSECONDS=' is stripped from Spectrum Title if present in .mgf or in search result.

Myrimatch

- Spectrum Title is corrected
- 15N label is not formatted correctly these modifications are removed for further analysis.
- When using 15N modifications on amino acids and Carbamidomethyl myrimatch reports sometimes Carboxymethylation on Cystein.

MS-GF+

- 15N label is not formatted correctly these modifications are removed for further analysis.
- 'Is decoy' column is properly set to true/false
- Carbamidomethyl is updated and set if label is 15N

OMSSA

- Carbamidomethyl is updated and set

MS-Amanda

- multiple protein ID per peptide are splitted in two entries. (is done in MS-Amanda postflight)

MSFragger

- 15N modification have to be removed from Modifications and the merged modifications have to be corrected.

pGlyco

- reformat modifications
- reformat glycan

Upeptide mapper v1_0_0

class ursgal.wrappers.upeptide_mapper_1_0_0.**upeptide_mapper_1_0_0**(**args*, ***kwargs*)

upeptide_mapper_1_0_0 UNode

Note: Different converter versions can be used (see parameter 'peptide_mapper_converter_version') as well as different classes inside the converter node (see parameter 'peptide_mapper_class_version')

Available converter classes of upeptide_mapper_1_0_0

- UPeptideMapper_v3 (default)
- UPeptideMapper_v4 (no buffering and enhanced speed to v3)
- UPeptideMapper_v2

_execute()

Peptides from search engine csv file are mapped to the given database(s)

ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_mapper

Peptide mapping implementation as Unode.

Parameters

- **input_file** (*str*) – input filename of csv
- **output_file** (*str*) – output filename
- **params** (*dict*) – dictionary containing ursgal params

Results and fixes

- All peptide Sequences are remapped to their corresponding protein, assuring correct start, stop, pre and post aminoacid.
- It is determined if the corresponding proteins are decoy proteins. These peptides are reported after the mapping process.
- Non-mappable peptides are reported. This can e.g. due to 'X' in protein sequences in the fasta file or other non-standard amino acids. These are sometimes replaced/interpreted/interpolated by the search engine. A recheck is performed if the peptides can be mapped containing an 'X' at any position. These peptides are also reported. If peptides can still not be mapped after re-mapping, these are reported as well.

Mapper class v4 (dev)

class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_

UPeptideMapper V4

Improved version of class version 3 (changes proposed by Christian)

Note: Uses the implementation of Aho-Corasick algorithm pyahocorasick. Please refer to https://pypi.python.org/pypi/pyahocorasick/ for more information.

---

cache_database(*fasta_database*)

Function to cache the given fasta database.

Parameters **fasta_database** (*str*) – path to the fasta database

---

Note: If the same fasta_name is buffered again all info is purged from the class.

---

map_peptides(*peptide_list*)

Function to map a given peptide list in one batch.

Parameters **peptide_list** (*list*) – list with peptides to be mapped

Returns

Dictionary containing peptides as keys and lists of protein mappings as values of the given fasta_name

Return type peptide_2_protein_mappings (dict)

Note: Based on the number of peptides the returned mapping dictionary can become very large.

Warning: The peptide to protein mapping is resetted if a new list o peptides is mapped to the same database (fasta_name).

Examples:

```
peptide_2_protein_mappings['PEPTIDE']  = [
    {
        'start' : 1,
        'end'   : 10,
        'pre'   : 'K',
        'post'  : 'D',
        'id'    : 'BSA'
    }
]
```

class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_

UPeptideMapper V3

New improved version which is faster and consumes less memory than earlier versions. Is the new default version for peptide mapping.

Note: Uses the implementation of Aho-Corasick algorithm pyahocorasick. Please refer to https://pypi.python.org/pypi/pyahocorasick/ for more information.

Warning: The new implementation is still in beta/testing phase. Please use, check and interpret accordingly

cache_database(*fasta_database*, *fasta_name*)

Function to cache the given fasta database.

Parameters

- **fasta_database** (*str*) – path to the fasta database
- **fasta_name** (*str*) – name of the database (e.g. os.path.basename(fasta_database))

Note: If the same fasta_name is buffered again all info is purged from the class.

---

map_peptides(*peptide_list*, *fasta_name*)

Function to map a given peptide list in one batch.

Parameters

- **peptide_list** (*list*) – list with peptides to be mapped
- **fasta_name** (*str*) – name of the database (e.g. os.path.basename(fasta_database))

Returns

Dictionary containing peptides as keys and lists of protein mappings as values of the given fasta_name

Return type peptide_2_protein_mappings (dict)

Note: Based on the number of peptides the returned mapping dictionary can become very large.

Warning: The peptide to protein mapping is resetted if a new list o peptides is mapped to the same database (fasta_name).

Examples:

```
peptide_2_protein_mappings['BSA1']['PEPTIDE']  = [
    {
        'start' : 1,
        'end'   : 10,
        'pre'   : 'K',
        'post'  : 'D',
        'id'    : 'BSA'
    }
]
```

purge_fasta_info(*fasta_name*)
Purges regular sequence lookup and fcache for a given fasta_name

class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_
UPeptideMapper class offers ultra fast peptide to sequence mapping using a fast cache, hereafter referred to fcache.

The fcache is build using the *build_lookup_from_file* or *build_lookup* functions. The fcache can be queried using the UPeptideMapper.map_peptide() function.

Note: This is the deprectaed version of the peptide mapper which can be used by setting the parameter 'peptide_mapper_class_version' to 'UPeptideMapper_v2'. Otherwise the new mapper class version ('UPeptideMapper_v3') is used as default.

_create_fcache(*id=None*, *seq=None*, *fasta_name=None*)
Updates the fast cache with a given sequence

_format_hit_dict(*seq*, *start*, *end*, *id*)
Creates a formated dictionary from a single mapping hit. At the same time evaluating pre and pos amino acids from the given sequence Final output looks for example like this:

```
{
    'start' : 12,
    'end'   : 18,
    'id'    : 'Protein Id passed to the function',
    'pre'   : 'A',
    'post'  : 'V',
}
```

Note: If the pre or post amino acids are N- or C-terminal, respectively, then the reported amino acid will be '-'

build_lookup(*fasta_name=None*, *fasta_stream=None*, *force=True*)
Builds the fast cache and regular sequence dict from a fasta stream

build_lookup_from_file (*path_to_fasta_file*, *force=True*)
Builds the fast cache and regular sequence dict from a fasta stream

return the internal fasta name, i.e. dirs stripped away from the path

map_peptide (*peptide=None*, *fasta_name=None*, *force_regex=False*)
Maps a peptide to a fasta database.

Returns a list of single hits which look for example like this:

```
{
    'start' : 12,
    'end'   : 18,
    'id'    : 'Protein Id passed to the function',
    'pre'   : 'A',
    'post'  : 'V',
}
```

map_peptides (*peptide_list*, *fasta_name=None*, *force_regex=False*)
Wrapper function to map a given peptide list in one batch.

Parameters

- **peptide_list** (*list*) – list with peptides to be mapped
- **fasta_name** (*str*) – name of the database

purge_fasta_info (*fasta_name*)
Purges regular sequence lookup and fcache for a given fasta_name

Quantification Engines

pyQms 1_0_0

ursgal.resources.platform_independent.arc_independent.pyqms_1_0_0.pyqms_1_0_0.**main**(*mzml_file=
*out-
*put_file=N
*pickle_nam
*ev-
*i-
*dence_files
*fixed_label
*molecules=
*rt_border_
*la-
*bel='14N',
*la-
*bel_percen
*min_charg
*max_charg
*ev-
*i-
*dence_scor
*ms_level=
*triv-
*ial_names=
*pyqms_par
*ver-
*bose=True

DOCSTRING.

FlashLFQ 1_1_1

class ursgal.wrappers.flash_lfq_1_1_1.**flash_lfq_1_1_1**(**args*, ***kwargs*)

postflight()
This can be/is overwritten by the engine uNode class

preflight()
This can be/is overwritten by the engine uNode class

Validation Engines

Kojak tailored Percolator 2_08

class ursgal.wrappers.kojak_percolator_2_08.**kojak_percolator_2_08**(**args*, ***kwargs*)

Kojak adjusted Percolator 2_08 UNode

Kojak provides preformatted Percolator input, this is used direclty as the input file for Percolator. In contrast to the original Percolator node, the input files are not reformatted or used to write a new input file.

Note: Percolator (2.08) has to be symlinked or copied to engine-folder 'kojak_percolator_2_08' in order to

make this node work.

Reference: Käll L, Canterbury JD, Weston J, Noble WS, MacCoss MJ. (2007) Semi-supervised learning for peptide identification from shotgun proteomics datasets.

postflight()
Convert the percolator output .tsv into the .csv format with headers as in the unified csv format.

preflight()
Formatting the command line to via self.params

Percolator

Available Percolator versions, starting with the newest version:

class ursgal.wrappers.percolator_3_4_0.**percolator_3_4_0**(**args*, ***kwargs*)
Percolator 3.4.0 UNode

q-value and posterior error probability calculation by a semi-supervised learning algorithm that dynamically learns to separate target from decoy peptide-spectrum matches (PSMs)

Reference: Matthew The, Michael J. MacCoss, William S. Noble, Lukas KÃ¤ll "Fast and Accurate Protein False Discovery Rates on Large-Scale Proteomics Data Sets with Percolator 3.0"

Noe: Please download Percolator corresponding to your OS from: https://github.com/percolator/percolator/releases

class ursgal.wrappers.percolator_3_2_1.**percolator_3_2_1**(**args*, ***kwargs*)
Percolator 3.2.1 UNode

q-value and posterior error probability calculation by a semi-supervised learning algorithm that dynamically learns to separate target from decoy peptide-spectrum matches (PSMs)

Reference: Matthew The, Michael J. MacCoss, William S. Noble, Lukas KÃ¤ll "Fast and Accurate Protein False Discovery Rates on Large-Scale Proteomics Data Sets with Percolator 3.0"

Noe: Please download Percolator corresponding to your OS from: https://github.com/percolator/percolator/releases

postflight()
read the output and merge in back to the ident csv

preflight()
Formating the command line to via self.params

class ursgal.wrappers.percolator_2_08.**percolator_2_08**(**args*, ***kwargs*)
Percolator 2_08 UNode

q-value and posterior error probability calculation by a semi-supervised learning algorithm that dynamically learns to separate target from decoy peptide-spectrum matches (PSMs)

Reference: Käll L, Canterbury JD, Weston J, Noble WS, MacCoss MJ. (2007) Semi-supervised learning for peptide identification from shotgun proteomics datasets.

postflight()
read the output and merge in back to the ident csv

preflight()
Formating the command line to via self.params

PTMiner 1.0

class ursgal.wrappers.ptminer_1_0.**ptminer_1_0**(**args*, ***kwargs*)
ptminer_1_0 used for mass shifts localization and peptides anotation

postflight(*anno_result=None*, *csv_input=None*, *merged_results_csv=None*)
This can be/is overwritten by the engine uNode class

preflight()
This can be/is overwritten by the engine uNode class

PTM-Sheperd 0.3.5

class ursgal.wrappers.ptmshepherd_0_3_5.**ptmshepherd_0_3_5**(**args*, ***kwargs*)
ptmshepherd_0_3_5 for validation and annotation of mass sifts from open modeification search results

postflight(*csv_input=None*)
Take rawlocalize and rawsimrt files for PSM annotation. Use global.profile for modification annotation. Merge result with original input file. Rename modsummary and global.profile to keep them as summary output files.

preflight()
Rewrite input file in tsv format. Write config file based on params. Generate command line from params.

write_input_tsv(*input_csv*, *tmp_dir*)
convert ursgal csv into PTM-Shepherd input tsv (same format as Philosopher output psms tsv)

pGlyco FDR 2.2.0

class ursgal.wrappers.pglyco_fdr_2_2_0.**pglyco_fdr_2_2_0**(**args*, ***kwargs*)
Unode for pGlycoFDR included in pGlyco 2.2.0 This node allows post-processing of pGlyco results

---

Note: Please download pGlycoFDR manually as part of pGlyco 2.2.0 https://github.com/pFindStudio/pGlyco2

Reference: Liu MQ, Zeng WF,, Fang P, Cao WQ, Liu C, Yan GQ, Zhang Y, Peng C, Wu JQ, Zhang XJ, Tu HJ, Chi H, Sun RX, Cao Y, Dong MQ, Jiang BY, Huang JM, Shen HL, Wong CCL, He SM, Yang PY. (2017) pGlyco 2.0 enables precision N-glycoproteomics with comprehensive quality control and one-step mass spectrometry for intact glycopeptide identification. Nat Commun 8(1)

postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

qvality 2_02

class ursgal.wrappers.qvality_2_02.**qvality_2_02**(**args*, ***kwargs*)
qvality_2_02 UNode

q-value and posterior error probability calculation from score distributions

Reference: Kll L, Storey JD, Noble WS (2009) QVALITY: non-parametric estimation of q-values and posterior error probabilities.

postflight()
Parse the qvality output and merge it back into the csv file

preflight()
Formating the command line to via self.params

Visualizer

Plot pyGCluster heatmap from CSV 1_0_0

class ursgal.wrappers.plot_pygcluster_heatmap_from_csv_1_0_0.**plot_pygcluster_heatmap_from_c**

plot_pygcluster_heatmap_from_csv_1_0_0 UNode

Venn Diagram v1_0_0

class ursgal.wrappers.venndiagram_1_0_0.**venndiagram_1_0_0**(**args*, ***kwargs*)
Venn Diagram uNode

_execute()

Plot Venn Diagramm for a list of .csv result files (2-5) Arguments are set in uparams.py but passed to the engine by self.params attribute Returns:

dict: results for the different areas e.g. dict['C-(A|B|D)']['results'] Output file is written to the common_top_level_dir

ursgal.resources.platform_independent.arc_independent.venndiagram_1_0_0.venndiagram_1_0_0.r

Creates a simple SVG VennDiagram requires 2, 3, 4 or 5 sets as arguments

Keyword Arguments

- **output_file** –
- **header** –
- **label_A** –
- **label_B** –
- **label_C** –
- **label_D** –
- **label_E** –
- **color_A** – e.g. #FF8C00
- **color_B** –
- **color_C** –
- **color_D** –
- **color_E** –
- **font** –

the function returns a dict with the following keys were the results can be accesse by e.g. dict['C-(A|B|D)']['results']

'A&B-(C|D)' 'C&D-(A|B)' 'B&C-(A|D)' 'A&B&C&D' 'A&C-(B|D)' 'B&D-(A|C)' 'A&D-(B|C)' '(A&C&D)-B' '(A&B&D)-C' '(A&B&C)-D' '(B&C&D)-A' 'A-(B|C|D)' 'D-(A|B|C)' 'B-(A|C|D)' 'C-(A|B|D)'

or for 2 or 3 or 5 VennDiagrams the appropriate combinations ...

6.2 How to extend and create new engines ?

6.2.1 Create/Implement your own UNode

Before implementing your own UNode, make sure that you have read about the *General structure* of Ursgal. This page will explain how to integrate a standalone executable or Python script into Ursgal's structure of **resources**, **wrappers** and **uparams.py**, based on two examples:

- A Python script: filter_csv_1_0_0.py
- A standalone search engine: MS-GF+ v9979

1. Integration into Resources

The **resources/** folder contains the main code of each UNode (an executable or Python script). This executable should be standalone and executable from the command line. Each UNode requires its own subfolder in the **resources/** folder, which contains the executable.

Note: The UNodes' **resources/** subfolder, **wrapper/** file and **wrapper/** file Python class should all have the same name (lowercase and underscores instead of spaces, e.g. 'msgfplus_v9979' or 'filter_csv_1_0_0').

1. Platfom dependent engines need to be placed according to the platform: darwin (OS X), linux or win32 (Windows 32 or 64 bit)

   - <ursgal_path>**/resources/**<platform>/<architecture>/<name_of_engine>/source (executable + potential additional files)

   Example: MS-GF+ on windows 64 bit:

   - <ursgal_path>**/resources/**win32/64bit/msgfplus_v9979/MSGFPlus.jar

2. Architecture independent engines, like Python scripts or Java packages should be placed in **/resources/**platform_independent/arc_independent/

   - <ursgal_path>**/resources/**platform_independent/arc_independent/<name_of_engine>/engine.py

   Example: filter_csv_1_0_0.py:

   - <ursgal_path>**/resources/**platform_independent/arc_independent/filter_csv_1_0_0/filter_csv_1_0_0.py

   Actually, MS-GF+ is platform independent as well (since it it based on Java) and can therefore also be placed in:

   - <ursgal_path>**/resources/**platform_independent/arc_independent/msgfplus_v9979/MSGFPlus.jar

2. Integration into uparams.py

Each parameter that is used by an engine needs to be included in the file <ursgal_path>/ursgal/uparams.py. This is a dictionary containing all parameters that are available in ursgal, its structure is explained *here*.

For every parameter that can be used by a new engine, it should be checked if a corresponding parameter is already present in uparams.py. If this is the case, the new engine (unode name) needs to be included in 'available_in_unode'. Furthermore, 'ukey_translation' needs to contain the utranslation_style that is defined in the engines *META_INFO* translating the ursgal parameter into the engine-specific parameter name. The parameter values can be translated in 'uvalue_translation' using the utranslation_style as well (only if a translation is necessary).

Example: include the parameter '-e' for MS-GF+

```
# -e defines the enzyme that has been used for digestion. This is called 'enzyme' in␣
↪ursgal.
'enzyme' : {
    # include msgfplus_v9979 in available_in_unode
    'available_in_unode' : [
        'xtandem_vengeance',
        'msgfplus_v9979',
    ],
    # default_value, description, trigger_rerun, utag and uvalue_type don't need to␣
↪be changed
    'default_value' : "trypsin",
    'description' :  ''' Enzyme: Rule of protein cleavage
        Possible cleavages are ... '''
    'trigger_rerun' : True,
    # Translate the ursgal parameter name ('enzyme') to the MS-GF+ parameter name ('-e
↪') using the translation style (msgfplus_style_1) in ukey_translation
    'ukey_translation' : {
        'msgfplus_style_1' : '-e',
        'xtandem_style_1' : 'protein, cleavage site',
    },
    # Translate the ursgal parameter values (e.g. 'trypsin') to the MS-GF+ parameter␣
↪value (e.g. '1') using the translation style (msgfplus_style_1) in uvalue_
↪translation
    'uvalue_translation' : {
        'msgfplus_style_1' : {
            'alpha_lp' : '8',
            'argc' : '6',
            'aspn' : '7',
            'chymotrypsin' : '2',
            'glutamyl_endopeptidase' : '5',
            'lysc' : '3',
            'lysn' : '4',
            'no_cleavage' : '9',
            'nonspecific' : '0',
            'trypsin' : '1',
        },
        'xtandem_style_1' : {
            'argc' : '[R]|{P}',
            'aspn' : '[X]|[D]',
            'chymotrypsin' : '[FMWY]|{P}',
            'chymotrypsin_p' : '[FMWY]|[X]',
            'clostripain' : '[R]|[X]',
            'cnbr' : '[M]|{P}',
            'elastase' : '[AGILV]|{P}',
            'formic_acid' : '[D]|{P}',
```

(continues on next page)

(continued from previous page)

```
            'gluc' : '[DE]|{P}',
            'gluc_bicarb' : '[E]|{P}',
            'iodosobenzoate' : '[W]|[X]',
            'lysc' : '[K]|{P}',
            'lysc_p' : '[K]|[X]',
            'lysn' : '[X]|[K]',
            'lysn_promisc' : '[X]|[AKRS]',
            'nonspecific' : '[X]|[X]',
            'pepsina' : '[FL]|[X]',
            'protein_endopeptidase' : '[P]|[X]',
            'staph_protease' : '[E]|[X]',
            'tca' : '[FMWY]|{P},[KR]|{P},[X]|[D]',
            'trypsin' : '[KR]|{P}',
            'trypsin_cnbr' : '[KR]|{P},[M]|{P}',
            'trypsin_gluc' : '[DEKR]|{P}',
            'trypsin_p' : '[RK]|[X]',
        },
```

If a parameter is not yet present in uparams.py, you can add a new parameter containing all necessary information (see *here*).

Example add write_unfiltered_results for filter_csv_1_0_0

```
'write_unfiltered_results' : {
    'edit_version' : 1.00,
    'available_in_unode' : [
        'filter_csv_1_0_0',
    ],
    'triggers_rerun' : True,
    'ukey_translation' : {
        'filter_csv_style_1' : 'write_unfiltered_results',
    },
    'utag' : [
        'conversion',
    ],
    'uvalue_translation' : {
    },
    'uvalue_type' : 'bool',
    'uvalue_option' : {
    },
    'default_value' : False,
    'description' : \
        'Writes rejected results if True',
},
```

After changing uparams.py, please run the tests, especially chk_format_node_param_test.py to check for errors.

3. Implementation of the wrapper class

Each UNode has to have a Python wrapper file located in:

- <ursgal_path>**/wrappers/** <unode_name>.py

The UNode has to inherit from the UNode class, which during initialization injects the node related data into the class.

The default structure of the UNode class has to be:

```
class my_unode_1_0_0(ursgal.UNode):

    META_INFO = {}

    def __init__(self, *args, **kwargs):
        super(my_unode_1_0_0, self).__init__(*args, **kwargs)

    def preflight(self):
        # code that should be run before the UNode is executed
        # e.g. writing a config file
        return

    def postflight(self):
        # code that should be run after the UNode is executed
        # e.g. formatting the output file
        return
```

where *my_unode_1_0_0* is the name of the UNode. The META_INFO is explained *here* and is available as attribute of each UNode. One can define *preflight()* and *postflight()* methods that will be executed by the uNode during preflight and postflight (= before execution of the main executable and after execution).

3.1 Implementation of an engine from a command line tool

For binary executable UNodes, one has to create a command line list (see subprocess) in the *preflight()* method. The command list is used to run the UNode's executable with the appropriate command line parameters. It should include the executable path of the engine (accessible via self.exe) and all relevant parameters, available via self.params, containing the original parameters and values. self.params['translations'] contains translated values for all node-related parameters. Furthermore self.params['translations']['_grouped_by_translated_key'] is a dictionary containing all node-related parameters and their corresponding ursgal parameters with the translated values.

The command list is stored in self.params['command_list']. This list should be constructed in the UNode class *preflight()* method like this:

```
def preflight(self):

    # retrieve the path of the input file:
    input_file = os.path.join(
        self.params['input_dir_path'],
        self.params['input_file']
    )

    # retrieve the auto-generated output file name:
    output_file = os.path.join(
        self.params['output_dir_path'],
        self.params['output_file'],
    )

    # format parameters and input/output file names into command list:
    self.params['command_list'] = [
        self.exe,
        '-o',
        output_file,
        '-i',
        input_file,
        '--some_parameter',
        '{some_param_in_ursgal}'.format(**self.params['translations']),
```

(continues on next page)

(continued from previous page)

```
        '--another_parameter',
        '{original_engine_parameter}'.format(**self.params['translations']['_grouped_
↪by_translated_key']),
    ]
```

After *preflight()*, Ursgal automatically passes the command_list to Python's built-in subprocess module:

```
proc = subprocess.Popen(
    self.params['command_list'],
    stdout = subprocess.PIPE,
)
```

After the execution procedure, the *postflight()* sequence is executed (if a postflight function was defined as part of the class), e.g.:

```
def postflight(self):
    '''
    Move the result files to the Kojak folder, since the output files can
    not be specified manually.
    '''
    # kojak_extensions = [
    #     '.kojak.txt',
    #     '.pep.xml',
    #     '.perc.inter.txt',
    #     '.perc.intra.txt',
    #     '.perc.loop.txt',
    #     '.perc.single.txt',
    # ]
    for extension in self.META_INFO['all_extensions']:
        org_path = os.path.join(
            self.params['input_dir_path'],
            '{0}{1}'.format(
                self.params['file_root'],
                extension
            )
        )
        new_path = os.path.join(
            self.params['output_dir_path'],
            '{0}_kojak_{1}{2}'.format(
                self.params['file_root'],
                self.META_INFO['version'],
                extension
            )
        )
        if os.path.exists(org_path):
            shutil.move(
                org_path,
                new_path
            )
```

Example: ursgal/engines/msgfplus_v9979.py

```
#!/usr/bin/env python
import ursgal
import os
```

(continues on next page)

(continued from previous page)

```
class msgfplus_v9979( ursgal.UNode ):
    """
    MSGF+ UNode
    Parameter options at https://bix-lab.ucsd.edu/pages/viewpage.action?
↪pageId=13533355

    Reference:
    Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ,␣
↪Pevzner PA. (2010) The Generating Function of CID, ETD, and CID/ETD Pairs of Tandem␣
↪Mass Spectra: Applications to Database Search.
    """
    META_INFO = {
        'edit_version'                : 1.00,
        'name'                        : 'MSGF+',
        'version'                     : 'v9979',
        'release_date'                : '2010-12-1',
        'engine_type' : {
            'protein_database_search_engine' : True,
        },
        'input_extensions'            : ['.mgf', '.mzML', '.mzXML', '.ms2', '.pkl', '.
↪dta.txt'],
        'output_extensions'           : ['.mzid'],
        'create_own_folder'           : True,
        'in_development'              : False,
        'include_in_git'              : False,
        'utranslation_style'          : 'msgfplus_style_1',
        'engine' : {
            'platform_independent' : {
                'arc_independent' : {
                    'exe'             : 'MSGFPlus.jar',
                    'url'             : 'http://proteomics.ucsd.edu/Software/MSGFPlus/
↪MSGFPlus.zip',
                    'zip_md5'         : '82a3e2204ff698e260ac9f89d3880b59',
                    'additional_exe' : [],
                },
            },
        },
        'citation' : \
            'Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, '\
            'Mohammed S, Heck AJ, Pevzner PA. (2010) The Generating Function '\
            'of CID, ETD, and CID/ETD Pairs of Tandem Mass Spectra: '\
            'Applications to Database Search.',
    }

    def __init__(self, *args, **kwargs):
        super(msgfplus_v9979, self).__init__(*args, **kwargs)
        pass

    def preflight( self ):
        '''
        Formatting the command line via self.params

        Modifications file will be created in the output folder

        Returns:
                dict: self.params
        '''
```

(continues on next page)

(continued from previous page)

```
        translations = self.params['translations']['_grouped_by_translated_key']

        self.params[ 'command_list' ] = [
            'java',
            '-jar',
            self.exe,
        ]

        self.params['translations']['mgf_input_file'] = os.path.join(
            self.params['input_dir_path'],
            self.params['input_file']
        )
        translations['-s']['mgf_input_file'] = self.params['translations']['mgf_input_
↪file']

        self.params['translations']['output_file_incl_path'] = os.path.join(
            self.params['output_dir_path'],
            self.params['output_file']
        )
        translations['-o']['output_file_incl_path'] = self.params['translations'][
↪'output_file_incl_path']

        self.params['translations']['modification_file'] = os.path.join(
            self.params['output_dir_path'],
            self.params['output_file'] + '_Mods.txt'
        )
        self.created_tmp_files.append( self.params['translations']['modification_file
↪'] )
        translations['-mod']['modifications'] = self.params['translations'][
↪'modification_file']

        mods_file = open( self.params['translations']['modification_file'], 'w',␣
↪encoding = 'UTF-8' )
        modifications = []

        print('NumMods={0}'.format(translations['NumMods']['max_num_mods']), file =␣
↪mods_file)

        if self.params['translations']['label'] == '15N':
            for aminoacid, N15_Diff in ursgal.ukb.DICT_15N_DIFF.items():
                existing = False
                for mod in self.params[ 'mods' ][ 'fix' ]:
                    if aminoacid == mod[ 'aa' ]:
                        mod[ 'mass' ] += N15_Diff
                        mod[ 'name' ] += '_15N_{0}'.format(aminoacid)
                        existing = True
                if existing == True:
                    continue
                else:
                    modifications.append( '{0},{1},fix,any,15N_{1}'.format( N15_Diff,␣
↪aminoacid ) )

        for t in [ 'fix', 'opt' ]:
            for mod in self.params[ 'mods' ][ t ]:
                modifications.append( '{0},{1},{2},{3},{4}'.format(mod[ 'mass' ],␣
↪mod[ 'aa' ], t, mod[ 'pos' ], mod[ 'name' ] ) )
```

(continues on next page)

(continued from previous page)

```
        for mod in modifications:
            print( mod, file = mods_file )

        mods_file.close()

        translations['-t'] = {
            '-t' : '{0}{1}, {2}{1}'.format(
                translations['-t']['precursor_mass_tolerance_minus'],
                translations['-t']['precursor_mass_tolerance_unit'],
                translations['-t']['precursor_mass_tolerance_plus'],
            )
        }

        command_dict = {}

        for translated_key, translation_dict in translations.items():
            if translated_key == '-Xmx':
                self.params[ 'command_list' ].insert(1,'{0}{1}'.format(
                    translated_key,
                    list(translation_dict.values())[0]
                ))
            elif translated_key in ['label', 'NumMods']:
                continue
            elif len(translation_dict) == 1:
                command_dict[translated_key] = str(list(translation_dict.values())[0])
            else:
                print('The translatd key ', translated_key, ' maps on more than one␣
↪ukey, but no special rules have been defined')
                print(translation_dict)
                exit(1)
        for k, v in command_dict.items():
            self.params[ 'command_list' ].extend((k, v))

        return self.params
```

3.2 Implementation of a UNode from Python code

Using sys.argv or the argparse module, any Python code can be executed like a command line tool. Thus, it is possible to include pure Python UNodes using the steps described above. For convenience, it is also possible to import the main function of a Python script using self.import_engine_as_python_function(). This function can then be directly executed by Ursgal, which makes it possible to include Python scripts that don't use argparse or sys.argv. To skip command line execution and run the main function of a Python script, one has to define the *_execute()* method of the UNode class. There are several pure Python UNodes in Ursgal, e.g. filter_csv_1_0_0.py, get_ftp_files_1_0_0.py and many others.

Example: ursgal/engines/filter_csv_1_0_0.py

```
#!/usr/bin/env python
import ursgal
import importlib
import os
import sys
import pickle
import shutil
```

(continues on next page)

(continued from previous page)

```
class filter_csv_1_0_0( ursgal.UNode ):
    """filter_csv_1_0_0 UNode"""
    def __init__(self, *args, **kwargs):
        super(filter_csv_1_0_0, self).__init__(*args, **kwargs)

    def _execute( self ):
        print('[ -ENGINE- ] Executing conversion ..')
        self.time_point(tag = 'execution')

        # import the main function from the UNode's python script
        filter_csv_main = self.import_engine_as_python_function()

        if self.params['output_file'].lower().endswith('.csv') is False:
            raise ValueError('Trying to filter a non-csv file.')

        # receive name of the input file so it can be passed to main function
        input_file  = os.path.join(
            self.params['input_dir_path'],
            self.params['input_file']
        )
        # receive auto-generated filename from UController
        output_file = os.path.join(
            self.params['output_dir_path'],
            self.params['output_file']
        )

        # Sometimes, engine-specific code is required! For instance,
        # filter_csv() can produce a second output file with the columns
        # that were removed:

        if self.params['translations']['write_unfiltered_results'] is False:

            output_file_unfiltered = None
        else:
            file_extension = self.meta_unodes[ self.engine ].META_INFO.get(
                'output_suffix',
                None
            )
            new_file_extension = self.meta_unodes[ self.engine ].META_INFO.get(
                'rejected_output_suffix',
                None
            )
            output_file_unfiltered = output_file.replace(
                file_extension,
                new_file_extension
            )
            shutil.copyfile(
                '{0}.u.json'.format(output_file),
                '{0}.u.json'.format(output_file_unfiltered)
            )
        # Engine-specific code ends here

        # Call the Python script's main() function using the information
        # we collected above:
        filter_csv_main(
            input_file     = input_file,
```

(continues on next page)

(continued from previous page)

```
            output_file     = output_file,
            filter_rules    = self.params['translations']['csv_filter_rules'],
            output_file_unfiltered = output_file_unfiltered,
        )

        self.print_execution_time(tag='execution')
        return output_file
```

CHAPTER 7

Examples

Ursgal comes with multiple example scripts which can be used to test its functionality. Example scripts can also be used as templates for your own scripts.

7.1 Example Scripts

This is a collection of example scripts for some of Ursgal's functionality. The simple example scripts are designed to get familiar with executing engines through Ursgal. Examles of complete workflows can be modified for different needs. Further example scripts can also be found in ~ursgalexample_scripts

7.1.1 Simple Example Scripts

Simple example search

simple_example_search.**main**()
Executes a search with OMSSA, XTandem and MS-GF+ on the BSA1.mzML input_file

usage: ./simple_example_search.py

Note: Myrimatch does not work with this file. To use MSAmanda on unix platforms, please install mono (http://www.mono-project.com/download)

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

import ursgal
import os
import sys
import shutil
```

(continues on next page)

(continued from previous page)

```
def main():
    """
    Executes a search with OMSSA, XTandem and MS-GF+ on the BSA1.mzML
    input_file

    usage:
        ./simple_example_search.py

    Note:
        Myrimatch does not work with this file.
        To use MSAmanda on unix platforms, please install mono
        (http://www.mono-project.com/download)

    """
    uc = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "database": os.path.join(os.pardir, "example_data", "BSA.fasta"),
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
            # 'peptide_mapper_class_version' : 'UPeptideMapper_v2',
        },
    )

    if sys.maxsize > 2 ** 32:
        xtandem = "xtandem_vengeance"
    else:
        xtandem = "xtandem_sledgehammer"

    engine_list = [
        "omssa",
        xtandem,
        "msgfplus_v2016_09_16",
    ]

    mzML_file = os.path.join(
        os.pardir, "example_data", "BSA_simple_example_search", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:
        uc.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        uc.params["http_output_folder"] = os.path.dirname(mzML_file)
        uc.fetch_file(
            engine="get_http_files_1_0_0",
        )
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)
```

(continues on next page)

(continued from previous page)

```
    unified_file_list = []

    for engine in engine_list:
        unified_search_result_file = uc.search(
            input_file=mzML_file, engine=engine, force=False
        )
        unified_file_list.append(unified_search_result_file)

    uc.visualize(
        input_files=unified_file_list,
        engine="venndiagram_1_1_0",
    )
    return


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

Simple example using combined fdr (or pep)

simple_combined_fdr_score.**main**()

Executes a search with 3 different search engines on an example file from the data from Barth et al. (The same file that is used in the XTandem version comparison example.)

usage: ./simple_combined_fdr_score.py

This is a simple example script to show how results from multiple search engines can be combined using the Combined FDR Score approach of Jones et al. (2009).

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

import ursgal
import os


def main():
    """
    Executes a search with 3 different search engines on an example file from the
    data from Barth et al. (The same file that is used in the XTandem version
    comparison example.)

    usage:
        ./simple_combined_fdr_score.py


    This is a simple example script to show how results from multiple search engines
    can be combined using the Combined FDR Score approach of Jones et al. (2009).
    """

    engine_list = [
        "omssa_2_1_9",
        "xtandem_piledriver",
        # 'myrimatch_2_1_138',
        "msgfplus_v9979",
    ]
```

(continues on next page)

(continued from previous page)

```
    params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "modifications": [],
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_include_ext": [
            "JB_FASP_pH8_2-3_28122012.mzML",
        ],
        "ftp_output_folder": os.path.join(
            os.pardir, "example_data", "xtandem_version_comparison"
        ),
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
    }

    if os.path.exists(params["ftp_output_folder"]) is False:
        os.mkdir(params["ftp_output_folder"])

    uc = ursgal.UController(profile="LTQ XL low res", params=params)
    mzML_file = os.path.join(params["ftp_output_folder"], params["ftp_include_ext
↪"][0])
    if os.path.exists(mzML_file) is False:
        uc.fetch_file(engine="get_ftp_files_1_0_0")
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    validated_files_list = []
    for engine in engine_list:

        unified_result_file = uc.search(
            input_file=mzML_file,
            engine=engine,
        )

        validated_file = uc.validate(
            input_file=unified_result_file,
            engine="percolator_2_08",
        )

        validated_files_list.append(validated_file)

    combined_results = uc.combine_search_results(
        input_files=validated_files_list,
        engine="combine_FDR_0_1",
        # use combine_pep_1_0_0 for combined PEP :)
    )
    print("\tCombined results can be found here:")
    print(combined_results)
    return
```

(continues on next page)

(continued from previous page)

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

Simple de novo search

simple_de_novo_search.**main**()
Executes a search with Novor and PepNovo on the BSA1.mzML input_file

usage: ./simple_de_novo_search.py

Note: PepNovo currently only works for Linux. Novor needs to be downloaded from http://rapidnovor.com/novor/standalone/ and stored at <ursgal_path>/resources/<platform>/<architecture>/novor_1_1beta

---

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

import ursgal
import os
import sys
import shutil


def main():
    """
    Executes a search with Novor and PepNovo on the BSA1.mzML
    input_file

    usage:
        ./simple_de_novo_search.py

    Note:
        PepNovo currently only works for Linux.
        Novor needs to be downloaded from http://rapidnovor.com/novor/standalone/ and
        stored at <ursgal_path>/resources/<platform>/<architecture>/novor_1_1beta

    """
    uc = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
        },
    )

    engine_list = [
        "novor_1_05",
        "pepnovo",
    ]
```

(continues on next page)

(continued from previous page)

```
    mzML_file = os.path.join(
        os.pardir, "example_data", "BSA_simple_de_novo_search", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:
        uc.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        uc.params["http_output_folder"] = os.path.dirname(mzML_file)
        uc.fetch_file(
            engine="get_http_files_1_0_0",
        )
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)

    unified_file_list = []

    for engine in engine_list:
        unified_search_result_file = uc.search(
            input_file=mzML_file, engine=engine, force=False
        )
        unified_file_list.append(unified_search_result_file)

    uc.visualize(
        input_files=unified_file_list,
        engine="venndiagram_1_1_0",
    )
    return


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

Simple crosslink search

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

import ursgal
import os
import sys


def main():
    """
    Simple crosslink search using Kojak and an example file from Barth et al.
    2014, please note that this is only testing if Kojak works.
    The sample represents no crosslink data in particular, but Kojak is used to
    map possible disulfide bonds.

    Parameters have not been optimized yet, please use this script as a template
    to use Kojak. Please note the different approach for executing Percolator.
```

(continues on next page)

(continued from previous page)

```
    Note:

        Please note that Kojak has to installed manually at the resources folder
        under engine-folder (kojak_1_5_3).
        Additinally Percolator (2.08) has to be symlinked or copied to
        engine-folder 'kojak_percolator_2_08'.

    Usage:
        ./simple_srosslink_search.py <path_2_chlamydomonas_reinhardtii_database>

    Note:

        The peptide -.GHYLNATAGTC[-34.00]EEMMK.- from Rubisco large subunit is
        detected with a predicted disulfide bond site at this position.
        This site is reported to have a disulfide bond in the Uniprot database.
        The modification (C +32, C -34) was set according to  'Tsai PL, Chen SF,
        Huang SY (2013) Mass spectrometry-based strategies for protein disulfide
        bond identification. Rev Anal Chem 32: 257-268'
        Please use the reference C. reinhardtii (TaxId 3055 ) proteome from
        Unipot.

    """

    params = {
        "database": sys.argv[1],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_include_ext": [
            "JB_FASP_pH8_2-3_28122012.mzML",
        ],
        "ftp_output_folder": os.path.join(
            os.pardir, "example_data", "simple_crosslink_search"
        ),
        "cross_link_definition": ["C C -2 test_if_kojak_runs"],
        "mono_link_definition": ["C 32", "C -34"],
        "modifications": [
            "M,opt,any,Oxidation",  # Met oxidation
        ],
        "precursor_min_mass": 500,
        "precursor_max_mass": 8000,
        "precursor_mass_tolerance_plus": 15,
        "precursor_mass_tolerance_minus": 15,
        "max_accounted_observed_peaks": 0,  # i.e. all
        "max_num_mods": 2,
    }

    if os.path.exists(params["ftp_output_folder"]) is False:
        os.mkdir(params["ftp_output_folder"])

    uc = ursgal.UController(profile="LTQ XL low res", params=params)
    mzML_file = os.path.join(params["ftp_output_folder"], params["ftp_include_ext
↪"][0])
    if os.path.exists(mzML_file) is False:
        uc.fetch_file(engine="get_ftp_files_1_0_0")
```

(continues on next page)

(continued from previous page)

```
    td_database_name = params["database"].replace(".fasta", "_target_decoy.fasta")
    if os.path.exists(td_database_name) is False:
        uc.execute_misc_engine(
            input_files=[params["database"]],
            engine="generate_target_decoy_1_0_0",
            output_file_name=td_database_name,
        )
    uc.params["database"] = td_database_name

    engine = "kojak_1_5_3"

    mgf_file = uc.convert(
        input_file=mzML_file,
        engine="mzml2mgf",
    )
    raw_result = uc.search_mgf(
        input_file=mgf_file,
        engine=engine,
        force=False,
    )
    for extension in uc.unodes[engine]["META_INFO"]["all_extensions"]:
        if "perc" not in extension:
            continue
        file_to_validate = raw_result.replace(
            uc.unodes[engine]["META_INFO"]["output_extension"], extension
        )
        try:
            uc.validate(file_to_validate, "kojak_percolator_2_08")
        except:
            pass

    return


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

Target decoy generation

target_decoy_generation_example.**main**()

Simple example script how to generate a target decoy database.

Note: By default a 'shuffled peptide preserving cleavage sites' database is generated. For this script a 'reverse protein' database is generated.

---

usage:

./target_decoy_generation_example.py

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

import ursgal
import os
```

(continues on next page)

(continued from previous page)

```
def main():
    """
    Simple example script how to generate a target decoy database.

    Note:
        By default a 'shuffled peptide preserving cleavage sites' database is
        generated. For this script a 'reverse protein' database is generated.

    usage:

        ./target_decoy_generation_example.py

    """
    params = {
        "enzyme": "trypsin",
        "decoy_generation_mode": "reverse_protein",
    }

    fasta_database_list = [os.path.join(os.pardir, "example_data", "BSA.fasta")]

    uc = ursgal.UController(params=params)

    new_target_decoy_db_name = uc.execute_misc_engine(
        input_file=fasta_database_list,
        engine="generate_target_decoy_1_0_0",
        output_file_name="my_BSA_target_decoy.fasta",
    )
    print("Generated target decoy database: {0}".format(new_target_decoy_db_name))


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

Convert .raw to .mzML

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

Convert a .raw file to .mzML using the ThermoRawFileParser. The given argument can be either a single file or a folder containing raw files.

Usage: ./convert_raw_to_mzml.py <raw_file/raw_file_folder>

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

import ursgal
import os
import sys
import glob


def main(input_path=None):
    """
    Convert a .raw file to .mzML using the ThermoRawFileParser.
```

(continues on next page)

(continued from previous page)

```
    The given argument can be either a single file or a folder
    containing raw files.

    Usage:
        ./convert_raw_to_mzml.py <raw_file/raw_file_folder>
    """
    R = ursgal.UController()

    # Check if single file or folder.
    # Collect raw files if folder is given
    input_file_list = []
    if input_path.lower().endswith(".raw"):
        input_file_list.append(input_path)
    else:
        for raw in glob.glob(os.path.join("{0}".format(input_path), "*.raw")):
            input_file_list.append(raw)

    # Convert raw file(s)
    for raw_file in input_file_list:
        mzml_file = R.convert(
            input_file=raw_file,
            engine="thermo_raw_file_parser_1_1_2",
        )


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(main.__doc__)
    main(input_path=sys.argv[1])
```

Simple mgf conversion

simple_mgf_conversion.**main**()

Simple example script to demonstrate conversion for mzML to mgf file conversion

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

import ursgal
import os
import shutil


def main():
    """
    Simple example script to demonstrate conversion for mzML to mgf file
    conversion

    """
    uc = ursgal.UController(
        profile="LTQ XL low res",
        params={},
    )

    mzML_file = os.path.join(
```

(continues on next page)

(continued from previous page)

```
        os.pardir, "example_data", "simple_mzml_to_mgf_conversion", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:
        uc.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        uc.params["http_output_folder"] = os.path.dirname(mzML_file)
        uc.fetch_file(engine="get_http_files_1_0_0")
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)

    mgf_file = uc.convert(
        input_file=mzML_file, engine="mzml2mgf_1_0_0"  # from OpenMS example files
    )


if __name__ == "__main__":
    print(__doc__)
    main()
```

Mgf conversion loop examples

mgf_conversion_inside_and_outside_loop.**main**()

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

import ursgal
import os
import shutil


def main():
    """"""
    R = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "database": os.path.join(os.pardir, "example_data", "BSA.fasta"),
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation[]
            ],
        },
    )

    engine = "omssa"
    output_files = []

    mzML_file = os.path.join(
        os.pardir, "example_data", "mgf_conversion_example", "BSA1.mzML"
    )
```

(continues on next page)

(continued from previous page)

```
    if os.path.exists(mzML_file) is False:
        R.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        R.params["http_output_folder"] = os.path.dirname(mzML_file)
        R.fetch_file(engine="get_http_files_1_0_0")
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)

    # First method: Convert to MGF outside of the loop:
    # (saves some time cause the MGF conversion is not always re-run)
    mgf_file = R.convert(
        input_file=mzML_file, engine="mzml2mgf_1_0_0"  # from OpenMS example files
    )
    for prefix in ["10ppm", "20ppm"]:
        R.params["prefix"] = prefix

        output_file = R.search(
            input_file=mgf_file,
            engine=engine,
            # output_file_name = 'some_userdefined_name'
        )

    # Second method: Automatically convert to MGF inside the loop:
    # (MGF conversion is re-run every time because the prexix changed!)
    for prefix in ["5ppm", "15ppm"]:
        R.params["prefix"] = prefix

        output_file = R.search(
            input_file=mzML_file,  # from OpenMS example files
            engine=engine,
            # output_file_name = 'another_fname',
        )
        output_files.append(output_file)

    print("\tOutput files:")
    for f in output_files:
        print(f)


if __name__ == "__main__":
    print(__doc__)
    main()
```

Simple Venn diagram

simple_venn_example.**main**()

Example for plotting a simple Venn diagram with single ursgal csv files.

usage: ./simple_venn_example.py

```
#!/usr/bin/env python3
```

(continues on next page)

(continued from previous page)

```
# encoding: utf-8

import ursgal
import os


def main():
    """
    Example for plotting a simple Venn diagram with single ursgal csv files.

    usage:
        ./simple_venn_example.py


    """
    uc = ursgal.UController(
        profile="LTQ XL low res",
        params={"visualization_label_positions": {"0": "omssa", "1": "xtandem"}},
    )

    file_list = [
        os.path.join(
            os.pardir, "tests", "data", "omssa_2_1_9", "test_BSA1_omssa_2_1_9.csv"
        ),
        os.path.join(
            os.pardir,
            "tests",
            "data",
            "xtandem_sledgehammer",
            "test_BSA1_xtandem_sledgehammer.csv",
        ),
    ]

    uc.visualize(
        input_files=file_list,
        engine="venndiagram_1_1_0",
        force=True,
    )
    return


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

Sanitize csv

sanitize_combined_results.**main**(*infile=None*)

Sanitize an Ursgal result file after combining results from multiple search engines (or single search engine results, with modified parameters)

usage: ./sanitize_combined_results.py <Ursgal_result_file>

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

(continues on next page)

(continued from previous page)

```
import sys
import os


def main(infile=None):
    """
    Sanitize an Ursgal result file after combining results
    from multiple search engines (or single search engine results,
    with modified parameters)

    usage:
        ./sanitize_combined_results.py <Ursgal_result_file>

    """
    mass_spectrometer = "QExactive+"
    params = {
        "precursor_mass_tolerance_minus": 10,
        "precursor_mass_tolerance_plus": 10,
        "frag_mass_tolerance": 10,
        "frag_mass_tolerance_unit": "ppm",
        "-xmx": "32g",
    }

    uc = ursgal.UController(profile=mass_spectrometer, params=params)

    # Parameters need to be adjusted based on the input file
    # and the desired type of sanitizing.
    # E.g. 'combined PEP' or 'Bayed PEP' can be used for combined PEP results,
    # while 'PEP' should be used for results from single engines.
    # A minimum difference between the top scoring, conflicting PSMs can be defined
    # using the parameters 'score_diff_threshold' and 'threshold_is_log10'
    uc.params.update(
        {
            "validation_score_field": "Bayes PEP",
            # 'validation_score_field': 'PEP',
            "bigger_scores_better": False,
            "num_compared_psms": 25,
            "accept_conflicting_psms": False,
            "threshold_is_log10": True,
            "score_diff_threshold": 0.0,
            "psm_defining_colnames": [
                "Spectrum Title",
                "Sequence",
                #     'Modifications',
                #     'Charge',
                #     'Is decoy',
            ],
            "max_num_psms_per_spec": 1,
            # 'preferred_engines': [],
            "preferred_engines": [
                "msfragger_2_3",
                "pipi_1_4_6",
                "moda_v1_61",
            ],
            "remove_redundant_psms": False,
        }
    )
```

(continues on next page)

(continued from previous page)

```
    sanitized_combined_results = uc.execute_misc_engine(
        input_file=infile,
        engine="sanitize_csv",
    )


if __name__ == "__main__":
    main(
        infile=sys.argv[1],
    )
```

Test node execution

test_node_execution.**main**()

Testscript for executing the test node, which also tests the run time determination function.

Usage: ./test_node_excution.py

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import glob
import os.path
import sys
import tempfile
import time


def main():
    """
    Testscript for executing the test node, which also tests the run time
    determination function.

    Usage:
        ./test_node_excution.py

    """
    uc = ursgal.UController(verbose=True)
    temp_int, temp_fpath = tempfile.mkstemp(prefix="ursgal_", suffix=".csv")
    temp_fobject = open(temp_fpath, "w")
    print("test 1,2,3", file=temp_fobject)
    temp_fobject.close()
    test_1 = uc.execute_unode(temp_fpath, "_test_node")
    test_2 = uc.execute_unode(test_1, "_test_node")


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

7.1.2 Complete Workflow Scripts

Do it all folder wide

do_it_all_folder_wide.**main**(*folder=None*, *profile=None*, *target_decoy_database=None*)

An example test script to search all mzML files which are present in the specified folder. The search is currently performed on 4 search engines and 2 validation engines.

The machine profile has to be specified as well as the target-decoy database.

usage:

./do_it_all_folder_wide.py <mzML_folder> <profile> <target_decoy_database>

Current profiles:

- 'QExactive+'
- 'LTQ XL low res'
- 'LTQ XL high res'

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import sys
import glob
import os


def main(folder=None, profile=None, target_decoy_database=None):
    """
    An example test script to search all mzML files which are present in the
    specified folder. The search is currently performed on 4 search engines
    and 2 validation engines.

    The machine profile has to be specified as well as the target-decoy
    database.

    usage:

        ./do_it_all_folder_wide.py <mzML_folder> <profile> <target_decoy_database>


    Current profiles:

        * 'QExactive+'
        * 'LTQ XL low res'
        * 'LTQ XL high res'

    """
    # define folder with mzML_files as sys.argv[1]
    mzML_files = []
    for mzml in glob.glob(os.path.join("{0}".format(folder), "*.mzML")):
        mzML_files.append(mzml)

    mass_spectrometer = profile

    # We specify all search engines and validation engines that we want to use in a␣
↪list
    # (version numbers might differ on windows or mac):
    search_engines = [
```

(continues on next page)

(continued from previous page)

```
    "omssa",
    "xtandem_vengeance",
    "msgfplus_v2016_09_16",
    # 'msamanda_1_0_0_6300',
    # 'myrimatch_2_1_138',
]

validation_engines = [
    "percolator_2_08",
    "qvality",
]

# Modifications that should be included in the search
all_mods = [
    "C,fix,any,Carbamidomethyl",
    "M,opt,any,Oxidation",
    # 'N,opt,any,Deamidated',
    # 'Q,opt,any,Deamidated',
    # 'E,opt,any,Methyl',
    # 'K,opt,any,Methyl',
    # 'R,opt,any,Methyl',
    "*,opt,Prot-N-term,Acetyl",
    # 'S,opt,any,Phospho',
    # 'T,opt,any,Phospho',
    # 'N,opt,any,HexNAc'
]

# Initializing the Ursgal UController class with
# our specified modifications and mass spectrometer
params = {
    "database": target_decoy_database,
    "modifications": all_mods,
    "csv_filter_rules": [
        ["Is decoy", "equals", "false"],
        ["PEP", "lte", 0.01],
    ],
}

uc = ursgal.UController(profile=mass_spectrometer, params=params)

# complete workflow:
# every spectrum file is searched with every search engine,
# results are validated (for each engine seperately),
# validated results are merged and filtered for targets and PEP <= 0.01.
# In the end, all filtered results from all spectrum files are merged
for validation_engine in validation_engines:
    result_files = []
    for spec_file in mzML_files:
        validated_results = []
        for search_engine in search_engines:
            unified_search_results = uc.search(
                input_file=spec_file,
                engine=search_engine,
            )
            validated_csv = uc.validate(
                input_file=unified_search_results,
                engine=validation_engine,
```

(continues on next page)

(continued from previous page)

```
                )
                validated_results.append(validated_csv)

            validated_results_from_all_engines = uc.execute_misc_engine(
                input_file=validated_results,
                engine="merge_csvs_1_0_0",
            )
            filtered_validated_results = uc.execute_misc_engine(
                input_file=validated_results_from_all_engines,
                engine="filter_csv_1_0_0",
            )
            result_files.append(filtered_validated_results)

        results_all_files = uc.execute_misc_engine(
            input_file=result_files,
            engine="merge_csvs_1_0_0",
        )


if __name__ == "__main__":
    if len(sys.argv) < 3:
        print(main.__doc__)
        sys.exit(1)
    main(
        folder=sys.argv[1],
        profile=sys.argv[2],
        target_decoy_database=sys.argv[3],
    )
```

Large scale data analysis

barth_et_al_large_scale.**main**(*folder*)

Example script for reproducing the data for figure 3

usage:

./barth_et_al_large_scale.py <folder>

The folder determines the target folder where the files will be downloaded

Chlamydomonas reinhardtii samples

Three biological replicates of 4 conditions (2_3, 2_4, 3_1, 4_1)

For more details on the samples please refer to Barth, J.; Bergner, S. V.; Jaeger, D.; Niehues, A.; Schulze, S.; Scholz, M.; Fufezan, C. The interplay of light and oxygen in the reactive oxygen stress response of Chlamydomonas reinhardtii dissected by quantitative mass spectrometry. MCP 2014, 13 (4), 969–989.

Merge all search results (per biological replicate and condition, on folder level) on engine level and validate via percolator.

'LTQ XL high res':

- repetition 1
- repetition 2

'LTQ XL low res':

- repetition 3

Database:

- Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta

Note: The database and the files will be automatically downloaded from our webpage and peptideatlas

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


import ursgal
import glob
import os.path
import sys


def main(folder):
    """

    Example script for reproducing the data for figure 3

    usage:

        ./barth_et_al_large_scale.py <folder>

    The folder determines the target folder where the files will be downloaded

    Chlamydomonas reinhardtii samples

    Three biological replicates of 4 conditions (2_3, 2_4, 3_1, 4_1)

    For more details on the samples please refer to
    Barth, J.; Bergner, S. V.; Jaeger, D.; Niehues, A.; Schulze, S.; Scholz,
    M.; Fufezan, C. The interplay of light and oxygen in the reactive oxygen
    stress response of Chlamydomonas reinhardtii dissected by quantitative mass
    spectrometry. MCP 2014, 13 (4), 969-989.

    Merge all search results (per biological replicate and condition, on folder
    level) on engine level and validate via percolator.

    'LTQ XL high res':

        * repetition 1
        * repetition 2

    'LTQ XL low res':

        * repetition 3

    Database:

        * Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta

    Note:

        The database and the files will be automatically downloaded from our
        webpage and peptideatlas
```

(continues on next page)

(continued from previous page)

```
    """

    input_params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "modifications": [
            "M,opt,any,Oxidation",
            "*,opt,Prot-N-term,Acetyl",  # N-Acetylation
        ],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_output_folder_root": folder,
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
    }

    uc = ursgal.UController(params=input_params)

    if os.path.exists(input_params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    output_folder_to_file_list = {
        ("rep1_sample_2_3", "LTQ XL high res"): [
            "CF_07062012_pH8_2_3A.mzML",
            "CF_13062012_pH3_2_3A.mzML",
            "CF_13062012_pH4_2_3A.mzML",
            "CF_13062012_pH5_2_3A.mzML",
            "CF_13062012_pH6_2_3A.mzML",
            "CF_13062012_pH11FT_2_3A.mzML",
        ],
        ("rep1_sample_2_4", "LTQ XL high res"): [
            "CF_07062012_pH8_2_4A.mzML",
            "CF_13062012_pH3_2_4A_120615113039.mzML",
            "CF_13062012_pH4_2_4A.mzML",
            "CF_13062012_pH5_2_4A.mzML",
            "CF_13062012_pH6_2_4A.mzML",
            "CF_13062012_pH11FT_2_4A.mzML",
        ],
        ("rep1_sample_3_1", "LTQ XL high res"): [
            "CF_12062012_pH8_1_3A.mzML",
            "CF_13062012_pH3_1_3A.mzML",
            "CF_13062012_pH4_1_3A.mzML",
            "CF_13062012_pH5_1_3A.mzML",
            "CF_13062012_pH6_1_3A.mzML",
            "CF_13062012_pH11FT_1_3A.mzML",
        ],
        ("rep1_sample_4_1", "LTQ XL high res"): [
            "CF_07062012_pH8_1_4A.mzML",
            "CF_13062012_pH3_1_4A.mzML",
            "CF_13062012_pH4_1_4A.mzML",
            "CF_13062012_pH5_1_4A.mzML",
            "CF_13062012_pH6_1_4A.mzML",
```

(continues on next page)

(continued from previous page)

```
            "CF_13062012_pH11FT_1_4A.mzML",
        ],
        ("rep2_sample_2_3", "LTQ XL high res"): [
            "JB_18072012_2-3_A_FT.mzML",
            "JB_18072012_2-3_A_pH3.mzML",
            "JB_18072012_2-3_A_pH4.mzML",
            "JB_18072012_2-3_A_pH5.mzML",
            "JB_18072012_2-3_A_pH6.mzML",
            "JB_18072012_2-3_A_pH8.mzML",
        ],
        ("rep2_sample_2_4", "LTQ XL high res"): [
            "JB_18072012_2-4_A_FT.mzML",
            "JB_18072012_2-4_A_pH3.mzML",
            "JB_18072012_2-4_A_pH4.mzML",
            "JB_18072012_2-4_A_pH5.mzML",
            "JB_18072012_2-4_A_pH6.mzML",
            "JB_18072012_2-4_A_pH8.mzML",
        ],
        ("rep2_sample_3_1", "LTQ XL high res"): [
            "JB_18072012_3-1_A_FT.mzML",
            "JB_18072012_3-1_A_pH3.mzML",
            "JB_18072012_3-1_A_pH4.mzML",
            "JB_18072012_3-1_A_pH5.mzML",
            "JB_18072012_3-1_A_pH6.mzML",
            "JB_18072012_3-1_A_pH8.mzML",
        ],
        ("rep2_sample_4_1", "LTQ XL high res"): [
            "JB_18072012_4-1_A_FT.mzML",
            "JB_18072012_4-1_A_pH3.mzML",
            "JB_18072012_4-1_A_pH4.mzML",
            "JB_18072012_4-1_A_pH5.mzML",
            "JB_18072012_4-1_A_pH6.mzML",
            "JB_18072012_4-1_A_pH8.mzML",
        ],
        ("rep3_sample_2_3", "LTQ XL low res"): [
            "JB_FASP_pH3_2-3_28122012.mzML",
            "JB_FASP_pH4_2-3_28122012.mzML",
            "JB_FASP_pH5_2-3_28122012.mzML",
            "JB_FASP_pH6_2-3_28122012.mzML",
            "JB_FASP_pH8_2-3_28122012.mzML",
            "JB_FASP_pH11-FT_2-3_28122012.mzML",
        ],
        ("rep3_sample_2_4", "LTQ XL low res"): [
            "JB_FASP_pH3_2-4_28122012.mzML",
            "JB_FASP_pH4_2-4_28122012.mzML",
            "JB_FASP_pH5_2-4_28122012.mzML",
            "JB_FASP_pH6_2-4_28122012.mzML",
            "JB_FASP_pH8_2-4_28122012.mzML",
            "JB_FASP_pH11-FT_2-4_28122012.mzML",
        ],
        ("rep3_sample_3_1", "LTQ XL low res"): [
            "JB_FASP_pH3_3-1_28122012.mzML",
            "JB_FASP_pH4_3-1_28122012.mzML",
            "JB_FASP_pH5_3-1_28122012.mzML",
            "JB_FASP_pH6_3-1_28122012.mzML",
            "JB_FASP_pH8_3-1_28122012.mzML",
            "JB_FASP_pH11-FT_3-1_28122012.mzML",
```

(continues on next page)

(continued from previous page)

```
        ],
        ("rep3_sample_4_1", "LTQ XL low res"): [
            "JB_FASP_pH3_4-1_28122012.mzML",
            "JB_FASP_pH4_4-1_28122012.mzML",
            "JB_FASP_pH5_4-1_28122012.mzML",
            "JB_FASP_pH6_4-1_28122012.mzML",
            "JB_FASP_pH8_4-1_28122012.mzML",
            "JB_FASP_pH11-FT_4-1_28122012_130121201449.mzML",
        ],
    }

    for (outfolder, profile), mzML_file_list in sorted(
        output_folder_to_file_list.items()
    ):
        uc.params["ftp_output_folder"] = os.path.join(
            input_params["ftp_output_folder_root"], outfolder
        )
        uc.params["ftp_include_ext"] = mzML_file_list

        if os.path.exists(uc.params["ftp_output_folder"]) is False:
            os.makedirs(uc.params["ftp_output_folder"])

        uc.fetch_file(engine="get_ftp_files_1_0_0")

    if os.path.exists(input_params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    search_engines = [
        "omssa_2_1_9",
        "xtandem_piledriver",
        "myrimatch_2_1_138",
        "msgfplus_v9979",
        "msamanda_1_0_0_5243",
    ]

    # This dict will be populated with the percolator-validated results
    # of each engine ( 3 replicates x4 conditions = 12 files each )
    percolator_results = {
        "omssa_2_1_9": [],
        "xtandem_piledriver": [],
        "msgfplus_v9979": [],
        "myrimatch_2_1_138": [],
        "msamanda_1_0_0_5243": [],
    }

    five_files_for_venn_diagram = []

    for search_engine in search_engines:

        # This list will collect all 12 result files for each engine,
        # after Percolator validation and filtering for PSMs with a
        # FDR <= 0.01
        filtered_results_of_engine = []
        for mzML_dir_ext, mass_spectrometer in output_folder_to_file_list.keys():
            # for mass_spectrometer, replicate_dir in replicates:
            # for condition_dir in conditions:
            uc.set_profile(mass_spectrometer)
```

(continues on next page)

(continued from previous page)

```
            mzML_dir = os.path.join(
                input_params["ftp_output_folder_root"], mzML_dir_ext
            )
            # i.e. /media/plan-f/mzML/Christian_Fufezan/ROS_Experiment_2012/Juni_2012/
↪2_3/Tech_A/
            # all files ending with .mzml in that directory will be used!

            unified_results_list = []
            for filename in glob.glob(os.path.join(mzML_dir, "*.mzML")):
                # print(filename)
                if filename.lower().endswith(".mzml"):
                    # print(filename)
                    unified_search_results = uc.search(
                        input_file=filename,
                        engine=search_engine,
                    )
                    unified_results_list.append(unified_search_results)

            # Merging results from the 6 pH-fractions:
            merged_unified = uc.execute_misc_engine(
                input_file=unified_results_list,
                engine="merge_csvs_1_0_0",
            )

            # Validation with Percolator:
            percolator_validated = uc.validate(
                input_file=merged_unified,
                engine="percolator_2_08",  # one could replace this with 'qvality'
            )
            percolator_results[search_engine].append(percolator_validated)

            # At this point, the analysis is finished. We got
            # Percolator-validated results for each of the 3
            # replicates and 12 conditions.

            # But let's see how well the five search engines
            # performed! To compare, we collect all PSMs with
            # an estimated FDR <= 0.01 for each engine, and
            # plot this information with the VennDiagram UNode.
            # We will also use the Combine FDR Score method
            # to combine the results from all five engines,
            # and increase the number of identified peptides.

    five_large_merged = []
    filtered_final_results = []

    # We will estimate the FDR for all 60 files
    # (5 engines x12 files) when using percolator PEPs as
    # quality score
    uc.params["validation_score_field"] = "PEP"
    uc.params["bigger_scores_better"] = False

    # To make obtain smaller CSV files (and make plotting
    # less RAM-intensive, we remove all decoys and PSMs above
    # 0.06 FDR
    uc.params["csv_filter_rules"] = [
```

(continues on next page)

(continued from previous page)

```
        ["estimated_FDR", "lte", 0.06],
        ["Is decoy", "equals", "false"],
    ]
    for engine, percolator_validated_list in percolator_results.items():

        # unfiltered files for cFDR script
        twelve_merged = uc.execute_misc_engine(
            input_file=percolator_validated_list,
            engine="merge_csvs_1_0_0",
        )

        twelve_filtered = []
        for one_of_12 in percolator_validated_list:
            one_of_12_FDR = uc.validate(
                input_file=one_of_12, engine="add_estimated_fdr_1_0_0"
            )
            one_of_12_FDR_filtered = uc.execute_misc_engine(
                input_file=one_of_12_FDR, engine="filter_csv_1_0_0"
            )
            twelve_filtered.append(one_of_12_FDR_filtered)

        # For the combined FDR scoring, we merge all 12 files:
        filtered_merged = uc.execute_misc_engine(
            input_file=twelve_filtered, engine="merge_csvs_1_0_0"
        )

        five_large_merged.append(twelve_merged)
        filtered_final_results.append(filtered_merged)

    # The five big merged files of each engine are combined:
    cFDR = uc.combine_search_results(
        input_files=five_large_merged,
        engine="combine_FDR_0_1",
    )

    # We estimate the FDR of this combined approach:
    uc.params["validation_score_field"] = "Combined FDR Score"
    uc.params["bigger_scores_better"] = False

    cFDR_FDR = uc.validate(input_file=cFDR, engine="add_estimated_fdr_1_0_0")

    # Removing decoys and low quality hits, to obtain a
    # smaller file:
    uc.params["csv_filter_rules"] = [
        ["estimated_FDR", "lte", 0.06],
        ["Is decoy", "equals", "false"],
    ]
    cFDR_filtered_results = uc.execute_misc_engine(
        input_file=cFDR_FDR,
        engine="filter_csv_1_0_0",
    )
    filtered_final_results.append(cFDR_filtered_results)

    # Since we produced quite a lot of files, let's print the full
    # paths to our most important result files so we find them quickly:
    print(
        """
```

(continues on next page)

(continued from previous page)

```
These files can now be easily parsed and plotted with your
plotting tool of choice! We used the Python plotting library
matplotlib. Each unique combination of Sequence, modification
and charge was counted as a unique peptide.
        """
    )
    print("\n########### Result files: ##############")
    for result_file in filtered_final_results:
        print("\t*{0}".format(result_file))


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(main.__doc__)
        sys.exit(1)
    main(sys.argv[1])
```

Complete workflow for human BR dataset analysis

human_br_complete_workflow.**main**(*folder*)

usage:

./human_br_complete_workflow.py <folder_with_human_br_files>

This scripts produces the data for figure 3.

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import os
import sys
import pprint


def main(folder):
    """

    usage:

        ./human_br_complete_workflow.py <folder_with_human_br_files>

    This scripts produces the data for figure 3.

    """

    # Initialize the UController:
    uc = ursgal.UController(
        params={
            "enzyme": "trypsin",
            "decoy_generation_mode": "reverse_protein",
        }
    )

    # MS Spectra, downloaded from http://proteomecentral.proteomexchange.org
    # via the dataset accession PXD000263 and converted to mzML
```

(continues on next page)

(continued from previous page)

```
    mass_spec_files = [
        "120813OTc1_NQL-AU-0314-LFQ-LCM-SG-01_013.mzML",
        "120813OTc1_NQL-AU-0314-LFQ-LCM-SG-02_025.mzML",
        "120813OTc1_NQL-AU-0314-LFQ-LCM-SG-03_033.mzML",
        "120813OTc1_NQL-AU-0314-LFQ-LCM-SG-04_048.mzML",
    ]

    for mass_spec_file in mass_spec_files:
        if os.path.exists(os.path.join(folder, mass_spec_file)) is False:
            print(
                "Please download RAW files to folder {} and convert to mzML:".format(
                    folder
                )
            )
            pprint.pprint(mass_spec_files)
            sys.exit(1)

    # mods from Wen et al. (2015):
    modifications = [
        # Carbamidomethyl  (C) was set as fixed modification
        "C,fix,any,Carbamidomethyl",
        "M,opt,any,Oxidation",  # Oxidation (M) as well as
        # Deamidated (NQ) were set as optional modification
        "N,opt,any,Deamidated",
        # Deamidated (NQ) were set as optional modification
        "Q,opt,any,Deamidated",
    ]

    # The target peptide database which will be searched (UniProt Human
    # reference proteome from July 2013)
    target_database = "uniprot_human_UP000005640_created_until_20130707.fasta"
    # Let's turn it into a target decoy database by reversing peptides:
    target_decoy_database = uc.execute_misc_engine(
        input_file=target_database, engine="generate_target_decoy_1_0_0"
    )

    # OMSSA parameters from Wen et al. (2015):
    omssa_params = {
        # (used by default) # -w
        "he": "1000",  # -he 1000
        "zcc": "1",  # -zcc 1
        "frag_mass_tolerance": "0.6",  # -to 0.6
        "frag_mass_tolerance_unit": "da",  # -to 0.6
        "precursor_mass_tolerance_minus": "10",  # -te 10
        "precursor_mass_tolerance_plus": "10",  # -te 10
        "precursor_mass_tolerance_unit": "ppm",  # -teppm
        "score_a_ions": False,  # -i 1,4
        "score_b_ions": True,  # -i 1,4
        "score_c_ions": False,  # -i 1,4
        "score_x_ions": False,  # -i 1,4
        "score_y_ions": True,  # -i 1,4
        "score_z_ions": False,  # -i 1,4
        "enzyme": "trypsin_p",  # -e 10
        "maximum_missed_cleavages": "1",  # -v 1
        "precursor_max_charge": "8",  # -zh 8
        "precursor_min_charge": "1",  # -zl 1
        "tez": "1",  # -tez 1
```

(continues on next page)

(continued from previous page)

```
        "precursor_isotope_range": "0,1",  # -ti 1
        "num_match_spec": "1",  # -hc 1
        "database": target_decoy_database,
        "modifications": modifications,
    }

    # MS-GF+ parameters from Wen et al. (2015):
    msgf_params = {
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_unit": "ppm",
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_minus": "10",
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_plus": "10",
        # the max number of optional modifications per peptide were set as 3
        # (used by default) # number of allowed isotope errors was set as 1
        "enzyme": "trypsin",  # the enzyme was set as trypsin
        # (used by default) # fully enzymatic peptides were specified, i.e. no non-
↪enzymatic termini
        "frag_method": "1",  # the fragmentation method selected in the search was CID
        "max_pep_length": "45",  # the maximum peptide length to consider was set as␣
↪45
        # the minimum precursor charge to consider if charges are not specified
        # in the spectrum file was set as 1
        "precursor_min_charge": "1",
        # the maximum precursor charge to consider was set as 8
        "precursor_max_charge": "8",
        # (used by default) # the parameter 'addFeatures' was set as 1 (required for␣
↪Percolator)
        # all of the other parameters were set as default
        # the instrument selected
        # was High-res
        "database": target_decoy_database,
        "modifications": modifications,
    }

    # X!Tandem parameters from Wen et al. (2015):
    xtandem_params = {
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_unit": "ppm",
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_minus": "10",
        # precursor ion mass tolerance was set to 10 ppm
        "precursor_mass_tolerance_plus": "10",
        # the fragment ion mass tolerance was set to 0.6 Da
        "frag_mass_tolerance": "0.6",
        # the fragment ion mass tolerance was set to 0.6 Da
        "frag_mass_tolerance_unit": "da",
        # parent monoisotopic mass isotope error was set as 'yes'
        "precursor_isotope_range": "0,1",
        "precursor_max_charge": "8",  # maximum parent charge of spectrum was set as 8
        "enzyme": "trypsin",  # the enzyme was set as trypsin ([RK]|[X])
        # the maximum missed cleavage sites were set as 1
        "maximum_missed_cleavages": "1",
        # (used by default) # no model refinement was employed.
        "database": target_decoy_database,
        "modifications": modifications,
```

(continues on next page)

(continued from previous page)

```
}

search_engine_settings = [
    # not used in Wen et al., so we use the same settings as xtandem
    ("msamanda_1_0_0_5243", xtandem_params, "LTQ XL high res"),
    # not used in Wen et al., so we use the same settings as xtandem
    ("myrimatch_2_1_138", xtandem_params, "LTQ XL high res"),
    # the instrument selected was High-res
    ("msgfplus_v9979", msgf_params, "LTQ XL high res"),
    ("xtandem_jackhammer", xtandem_params, None),
    ("omssa_2_1_9", omssa_params, None),
]

merged_validated_files_3_engines = []
merged_validated_files_5_engines = []

for engine, wen_params, instrument in search_engine_settings:

    # Initializing the uPLANIT UController class with
    # our specified modifications and mass spectrometer
    uc = ursgal.UController(params=wen_params)

    if instrument is not None:
        uc.set_profile(instrument)

    unified_results = []
    percolator_validated_results = []

    for mzML_file in mass_spec_files:
        unified_search_results = uc.search(
            input_file=mzML_file,
            engine=engine,
        )
        unified_results.append(unified_search_results)
        validated_csv = uc.validate(
            input_file=unified_search_results,
            engine="percolator_2_08",
        )
        percolator_validated_results.append(validated_csv)

    merged_validated_csv = uc.execute_misc_engine(
        input_file=percolator_validated_results, engine="merge_csvs_1_0_0"
    )
    merged_unvalidated_csv = uc.execute_misc_engine(
        input_file=unified_results,
        engine="merge_csvs_1_0_0",
    )

    if engine in ["omssa_2_1_9", "xtandem_jackhammer", "msgfplus_v9979"]:
        merged_validated_files_3_engines.append(merged_validated_csv)
    merged_validated_files_5_engines.append(merged_validated_csv)

uc.params["prefix"] = "5-engines-summary"
uc.combine_search_results(
    input_files=merged_validated_files_5_engines,
    engine="combine_FDR_0_1",
)
```

(continues on next page)

(continued from previous page)

```
    uc.params["prefix"] = "3-engines-summary"
    uc.combine_search_results(
        input_files=merged_validated_files_3_engines,
        engine="combine_FDR_0_1",
    )


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(main.__doc__)
        sys.exit(1)
    main(sys.argv[1])
```

Complete workflow for analysis with pGlyco

example_pglyco_workflow.**main**(*input_raw_file=None*, *database=None*, *enzyme=None*)

This script executes three steps:

- convert .raw file using pParse
- search the resulting .mgf file using pGlyco
- validate results using pGlycoFDR

Since pGlyco does not support providing a custom target-decoy database, a database including only target sequences must be provided. In this database, the N-glycosylation motif N-X-S/T needs to be replaced with J-X-S/T, which will be done by the pGlyco wrapper.

usage: ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

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

import ursgal
import os
import sys
import shutil


def main(input_raw_file=None, database=None, enzyme=None):
    """
    This script executes three steps:
        - convert .raw file using pParse
        - search the resulting .mgf file using pGlyco
        - validate results using pGlycoFDR

    Since pGlyco does not support providing a custom target-decoy database,
    a database including only target sequences must be provided.
    In this database, the N-glycosylation motif N-X-S/T needs to be replaced
    with J-X-S/T, which will be done by the pGlyco wrapper.

    usage:
        ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

    """
```

(continues on next page)

(continued from previous page)

```
    uc = ursgal.UController(
        profile="QExactive+",
        params={
            "database": database,
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
            "peptide_mapper_class_version": "UPeptideMapper_v4",
            "enzyme": enzyme,
            "frag_mass_tolerance": 20,
            "frag_mass_tolerance_unit": "ppm",
            "precursor_mass_tolerance_plus": 5,
            "precursor_mass_tolerance_minus": 5,
            "aa_exception_dict": {},
            "csv_filter_rules": [
                ["Is decoy", "equals", "false"],
                ["q-value", "lte", 0.01],
                ["Conflicting uparam", "contains_not", "enzyme"],
            ],
        },
    )

    xtracted_file = uc.convert(
        input_file=input_raw_file,
        engine="pparse_2_2_1",
        # force = True,
    )

    mzml_file = uc.convert(
        input_file=input_raw_file,
        engine="thermo_raw_file_parser_1_1_2",
    )

    # os.remove(mzml_file.replace('.mzML', '.mgf'))
    # os.remove(mzml_file.replace('.mzML', '.mgf.u.json'))
    mgf_file = uc.convert(
        input_file=mzml_file,
        engine="mzml2mgf_2_0_0",
        force=True,
    )

    search_result = uc.search_mgf(
        input_file=xtracted_file,
        engine="pglyco_db_2_2_2",
    )

    mapped_results = uc.execute_misc_engine(
        input_file=search_result,
        engine="upeptide_mapper",
    )

    unified_search_results = uc.execute_misc_engine(
        input_file=mapped_results, engine="unify_csv"
    )
```

(continues on next page)

(continued from previous page)

```
    validated_file = uc.validate(
        input_file=search_result,
        engine="pglyco_fdr_2_2_0",
    )

    mapped_validated_results = uc.execute_misc_engine(
        input_file=validated_file,
        engine="upeptide_mapper",
    )

    unified_validated_results = uc.execute_misc_engine(
        input_file=mapped_validated_results, engine="unify_csv"
    )

    filtered_validated_results = uc.execute_misc_engine(
        input_file=unified_validated_results,
        engine="filter_csv",
    )

    return


if __name__ == "__main__":
    main(input_raw_file=sys.argv[1], database=sys.argv[2], enzyme=sys.argv[3])
```

Complete workflow for analysis with TagGraph

example_pglyco_workflow.**main**(*input_raw_file=None*, *database=None*, *enzyme=None*)

This script executes three steps:

- convert .raw file using pParse
- search the resulting .mgf file using pGlyco
- validate results using pGlycoFDR

Since pGlyco does not support providing a custom target-decoy database, a database including only target sequences must be provided. In this database, the N-glycosylation motif N-X-S/T needs to be replaced with J-X-S/T, which will be done by the pGlyco wrapper.

usage: ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

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

import ursgal
import os
import sys
import shutil


def main(input_raw_file=None, database=None, enzyme=None):
    """
    This script executes three steps:
        - convert .raw file using pParse
        - search the resulting .mgf file using pGlyco
```

(continues on next page)

(continued from previous page)

```
    - validate results using pGlycoFDR

Since pGlyco does not support providing a custom target-decoy database,
a database including only target sequences must be provided.
In this database, the N-glycosylation motif N-X-S/T needs to be replaced
with J-X-S/T, which will be done by the pGlyco wrapper.

usage:
    ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

"""
uc = ursgal.UController(
    profile="QExactive+",
    params={
        "database": database,
        "modifications": [
            "M,opt,any,Oxidation",  # Met oxidation
            "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
            "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
        ],
        "peptide_mapper_class_version": "UPeptideMapper_v4",
        "enzyme": enzyme,
        "frag_mass_tolerance": 20,
        "frag_mass_tolerance_unit": "ppm",
        "precursor_mass_tolerance_plus": 5,
        "precursor_mass_tolerance_minus": 5,
        "aa_exception_dict": {},
        "csv_filter_rules": [
            ["Is decoy", "equals", "false"],
            ["q-value", "lte", 0.01],
            ["Conflicting uparam", "contains_not", "enzyme"],
        ],
    },
)

xtracted_file = uc.convert(
    input_file=input_raw_file,
    engine="pparse_2_2_1",
    # force = True,
)

mzml_file = uc.convert(
    input_file=input_raw_file,
    engine="thermo_raw_file_parser_1_1_2",
)

# os.remove(mzml_file.replace('.mzML', '.mgf'))
# os.remove(mzml_file.replace('.mzML', '.mgf.u.json'))
mgf_file = uc.convert(
    input_file=mzml_file,
    engine="mzml2mgf_2_0_0",
    force=True,
)

search_result = uc.search_mgf(
    input_file=xtracted_file,
    engine="pglyco_db_2_2_2",
```

(continues on next page)

(continued from previous page)

```
    )

    mapped_results = uc.execute_misc_engine(
        input_file=search_result,
        engine="upeptide_mapper",
    )

    unified_search_results = uc.execute_misc_engine(
        input_file=mapped_results, engine="unify_csv"
    )

    validated_file = uc.validate(
        input_file=search_result,
        engine="pglyco_fdr_2_2_0",
    )

    mapped_validated_results = uc.execute_misc_engine(
        input_file=validated_file,
        engine="upeptide_mapper",
    )

    unified_validated_results = uc.execute_misc_engine(
        input_file=mapped_validated_results, engine="unify_csv"
    )

    filtered_validated_results = uc.execute_misc_engine(
        input_file=unified_validated_results,
        engine="filter_csv",
    )

    return


if __name__ == "__main__":
    main(input_raw_file=sys.argv[1], database=sys.argv[2], enzyme=sys.argv[3])
```

Example search for 15N labeling and no label

search_with_label_15N.**main**()

Executes a search with 3 different search engines on an example file from the data from Barth et al. Two searches are performed pe engine for each 14N (unlabeled) and 15N labeling. The overlap of identified peptides for the 14N and 15N searches between the engines is visualized as well as the overlap between all 14N and 15N identified peptides.

usage: ./search_with_label_15N.py

Note: It is important to convert the mgf file outside of (and before :)) the search loop to avoid mgf file redundancy and to assure correct retention time mapping in the unify_csv node.

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

import ursgal
```

(continues on next page)

(continued from previous page)

```
import os


def main():
    """
    Executes a search with 3 different search engines on an example file from
    the data from Barth et al. Two searches are performed pe engine for each
    14N (unlabeled) and 15N labeling. The overlap of identified peptides
    for the 14N and 15N searches between the engines is visualized as well as
    the overlap between all 14N and 15N identified peptides.

    usage:
        ./search_with_label_15N.py

    Note:
        It is important to convert the mgf file outside of (and before :) ) the
        search loop to avoid mgf file redundancy and to assure correct retention
        time mapping in the unify_csv node.

    """

    engine_list = [
        "omssa_2_1_9",
        "xtandem_piledriver",
        "msgfplus_v9979",
    ]

    params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "modifications": [],
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_include_ext": [
            "JB_FASP_pH8_2-3_28122012.mzML",
        ],
        "ftp_output_folder": os.path.join(
            os.pardir, "example_data", "search_with_label_15N"
        ),
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
    }

    if os.path.exists(params["ftp_output_folder"]) is False:
        os.mkdir(params["ftp_output_folder"])

    uc = ursgal.UController(profile="LTQ XL low res", params=params)
    mzML_file = os.path.join(params["ftp_output_folder"], params["ftp_include_ext
↪"][0])
    if os.path.exists(mzML_file) is False:
        uc.fetch_file(engine="get_ftp_files_1_0_0")
```

(continues on next page)

(continued from previous page)

```
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    mgf_file = uc.convert(
        input_file=mzML_file,
        engine="mzml2mgf_1_0_0",
    )
    files_2_merge = {}
    label_list = ["14N", "15N"]
    for label in label_list:
        validated_and_filtered_files_list = []
        uc.params["label"] = label
        uc.params["prefix"] = label
        for engine in engine_list:
            search_result = uc.search_mgf(
                input_file=mgf_file,
                engine=engine,
            )
            converted_result = uc.convert(
                input_file=search_result,
                guess_engine=True,
            )
            mapped_results = uc.execute_misc_engine(
                input_file=converted_result,
                engine="upeptide_mapper",
            )
            unified_search_results = uc.execute_misc_engine(
                input_file=mapped_results, engine="unify_csv"
            )
            validated_file = uc.validate(
                input_file=unified_search_results,
                engine="percolator_2_08",
            )
            filtered_file = uc.execute_misc_engine(
                input_file=validated_file, engine="filter_csv"
            )
            validated_and_filtered_files_list.append(filtered_file)
        files_2_merge[label] = validated_and_filtered_files_list
        uc.visualize(
            input_files=validated_and_filtered_files_list,
            engine="venndiagram_1_1_0",
        )
    uc.params["prefix"] = None
    uc.params["label"] = ""
    uc.params["visualization_label_positions"] = {}
    label_comparison_file_list = []
    for n, label in enumerate(label_list):
        uc.params["visualization_label_positions"][str(n)] = label
        label_comparison_file_list.append(
            uc.execute_misc_engine(input_file=files_2_merge[label], engine="merge_csvs
↪")
        )
    uc.visualize(
        input_files=label_comparison_file_list,
        engine="venndiagram_1_1_0",
    )
```

(continues on next page)

(continued from previous page)

```
    return


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

7.1.3 Open Modification Search Scripts

Open modification search with three engines and combined PEP

open_modification_search_incl_combined_pep.**main**(*folder=None*, *database=None*, *enzyme=None*)

Example workflow to perform a open modification search with three independent search engines across all mzML files of a given folder and to statistically post-process and combine the results of all searches.

Usage: ./open_modification_search_incl_combined_pep.py <mzML_folder> <database> <enzyme>

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import sys
import glob
import os
import pprint


def main(folder=None, database=None, enzyme=None):
    """
    Example workflow to perform a open modification search with three independent␣
↪search engines
    across all mzML files of a given folder and to statistically post-process and␣
↪combine the
    results of all searches.

    Usage:
        ./open_modification_search_incl_combined_pep.py <mzML_folder> <database>
↪<enzyme>
    """
    # For this particular dataset, two enzymes were used, namely gluc and trypsin.
    mzml_files = []
    for mzml in glob.glob(os.path.join(folder, "*.mzML")):
        mzml_files.append(mzml)

    mass_spectrometer = "QExactive+"
    validation_engine = "percolator_3_4_0"
    search_engines = ["msfragger_2_3", "pipi_1_4_6", "moda_v1_61"]

    params = {
        "modifications": ["C,fix,any,Carbamidomethyl"],
        "csv_filter_rules": [
            ["Is decoy", "equals", "false"],
            ["PEP", "lte", 0.01],
        ],
        "frag_mass_tolerance_unit": "ppm",
        "frag_mass_tolerance": 20,
```

(continues on next page)

(continued from previous page)

```
        "precursor_mass_tolerance_unit": "ppm",
        "precursor_mass_tolerance_plus": 5,
        "precursor_mass_tolerance_minus": 5,
        "moda_high_res": False,
        "max_mod_size": 4000,
        "min_mod_size": -200,
        "precursor_true_units": "ppm",
        "precursor_true_tolerance": 5,
        "percolator_post_processing": "mix-max",
        "psm_defining_colnames": [
            "Spectrum Title",
            "Sequence",
            "Modifications",
            "Charge",
            "Is decoy",
            "Mass Difference",
        ],
        "database": database,
        "enzyme": enzyme,
    }

    uc = ursgal.UController(
        profile=mass_spectrometer,
        params=params,
    )

    # This will hold input to combined PEP engine
    combined_pep_input = defaultdict(list)

    # This dictionary will help organize which results to merge
    all_merged_results = defaultdict(list)

    for search_engine in search_engines:

        # The modification size for MSFragger is configured through precursor mass␣
↪tolerance
        if search_engine == "msfragger_2_3":
            uc.params.update(
                {
                    "precursor_mass_tolerance_unit": "da",
                    "precursor_mass_tolerance_plus": 4000,
                    "precursor_mass_tolerance_minus": 200,
                }
            )

        for n, spec_file in enumerate(mzml_files):
            # 1. convert to MGF
            mgf_file = uc.convert(
                input_file=spec_file,
                engine="mzml2mgf_2_0_0",
            )

            # 2. do the actual search
            raw_search_results = uc.search_mgf(
                input_file=mgf_file,
                engine=search_engine,
            )
```

(continues on next page)

(continued from previous page)

```
            # reset precursor mass tolerance just in case it was previously changed
            uc.params.update(
                {
                    "precursor_mass_tolerance_unit": "ppm",
                    "precursor_mass_tolerance_plus": 5,
                    "precursor_mass_tolerance_minus": 5,
                }
            )

            # 3. convert files to csv
            csv_search_results = uc.convert(
                input_file=raw_search_results,
                engine=None,
                guess_engine=True,
            )

            # 4. protein mapping.
            mapped_csv_search_results = uc.execute_misc_engine(
                input_file=csv_search_results,
                engine="upeptide_mapper_1_0_0",
            )

            # 5. Convert csv to unified ursgal csv format:
            unified_search_results = uc.execute_misc_engine(
                input_file=mapped_csv_search_results,
                engine="unify_csv_1_0_0",
                merge_duplicates=False,
            )

            # 6. Validate the results
            validated_csv = uc.validate(
                input_file=unified_search_results,
                engine=validation_engine,
            )

            # save the validated input for combined pep
            # Eventually, each sample will have 3 files correpsonding to the 3 search␣
↪engines
            combined_pep_input["sample_{0}".format(n)].append(validated_csv)

            filtered_validated_results = uc.execute_misc_engine(
                input_file=validated_csv,
                engine="filter_csv_1_0_0",
                merge_duplicates=False,
            )

            all_merged_results["percolator_only"].append(filtered_validated_results)

    # combined pep
    uc.params.update(
        {
            "csv_filter_rules": [
                ["Is decoy", "equals", "false"],
                ["combined PEP", "lte", 0.01],
            ],
            "psm_defining_colnames": [
```

(continues on next page)

(continued from previous page)

```
                "Spectrum Title",
                "Sequence",
                "Modifications",
                "Charge",
                "Is decoy",
            ],
        }
    )
    for sample in combined_pep_input.keys():
        combine_results = uc.execute_misc_engine(
            input_file=combined_pep_input[sample],
            engine="combine_pep_1_0_0",
        )

        filtered_validated_results = uc.execute_misc_engine(
            input_file=combine_results,
            engine="filter_csv_1_0_0",
        )
        all_merged_results["combined_pep"].append(filtered_validated_results)

    # separately merge results from the two types of validation techniques
    # We also add back "Mass Difference" to columns defining a PSM to avoid merging␣
↪mass differences
    uc.params.update(
        {
            "psm_defining_colnames": [
                "Spectrum Title",
                "Sequence",
                "Modifications",
                "Charge",
                "Is decoy",
                "Mass Difference",
            ],
        }
    )

    for validation_type in all_merged_results.keys():
        if validation_type == "percolator_only":
            uc.params["psm_colnames_to_merge_multiple_values"] = {
                "PEP": "min_value",
            }
        else:
            uc.params["psm_colnames_to_merge_multiple_values"] = {
                "combined PEP": "min_value",
                "Bayes PEP": "min_value",
            }

        uc.params["prefix"] = "All_{0}".format(
            validation_type
        )  # helps recognize files easily

        merged_results_one_rep = uc.execute_misc_engine(
            input_file=all_merged_results[validation_type],
            engine="merge_csvs_1_0_0",
            merge_duplicates=True,
        )
        uc.params["prefix"] = ""
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    main(
        folder=sys.argv[1],
        database=sys.argv[2],
        enzyme=sys.argv[3],
    )
```

Complete workflow for analysis with TagGraph

example_pglyco_workflow.**main**(*input_raw_file=None*, *database=None*, *enzyme=None*)

This script executes three steps:

- convert .raw file using pParse
- search the resulting .mgf file using pGlyco
- validate results using pGlycoFDR

Since pGlyco does not support providing a custom target-decoy database, a database including only target sequences must be provided. In this database, the N-glycosylation motif N-X-S/T needs to be replaced with J-X-S/T, which will be done by the pGlyco wrapper.

usage: ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

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

import ursgal
import os
import sys
import shutil


def main(input_raw_file=None, database=None, enzyme=None):
    """
    This script executes three steps:
        - convert .raw file using pParse
        - search the resulting .mgf file using pGlyco
        - validate results using pGlycoFDR

    Since pGlyco does not support providing a custom target-decoy database,
    a database including only target sequences must be provided.
    In this database, the N-glycosylation motif N-X-S/T needs to be replaced
    with J-X-S/T, which will be done by the pGlyco wrapper.

    usage:
        ./simple_example_pglyco_workflow.py <input_raw_file> <database> <enzyme>

    """
    uc = ursgal.UController(
        profile="QExactive+",
        params={
            "database": database,
            "modifications": [
```

(continues on next page)

(continued from previous page)

```
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
            "peptide_mapper_class_version": "UPeptideMapper_v4",
            "enzyme": enzyme,
            "frag_mass_tolerance": 20,
            "frag_mass_tolerance_unit": "ppm",
            "precursor_mass_tolerance_plus": 5,
            "precursor_mass_tolerance_minus": 5,
            "aa_exception_dict": {},
            "csv_filter_rules": [
                ["Is decoy", "equals", "false"],
                ["q-value", "lte", 0.01],
                ["Conflicting uparam", "contains_not", "enzyme"],
            ],
        },
    )

    xtracted_file = uc.convert(
        input_file=input_raw_file,
        engine="pparse_2_2_1",
        # force = True,
    )

    mzml_file = uc.convert(
        input_file=input_raw_file,
        engine="thermo_raw_file_parser_1_1_2",
    )

    # os.remove(mzml_file.replace('.mzML', '.mgf'))
    # os.remove(mzml_file.replace('.mzML', '.mgf.u.json'))
    mgf_file = uc.convert(
        input_file=mzml_file,
        engine="mzml2mgf_2_0_0",
        force=True,
    )

    search_result = uc.search_mgf(
        input_file=xtracted_file,
        engine="pglyco_db_2_2_2",
    )

    mapped_results = uc.execute_misc_engine(
        input_file=search_result,
        engine="upeptide_mapper",
    )

    unified_search_results = uc.execute_misc_engine(
        input_file=mapped_results, engine="unify_csv"
    )

    validated_file = uc.validate(
        input_file=search_result,
        engine="pglyco_fdr_2_2_0",
    )
```

(continues on next page)

(continued from previous page)

```
    mapped_validated_results = uc.execute_misc_engine(
        input_file=validated_file,
        engine="upeptide_mapper",
    )

    unified_validated_results = uc.execute_misc_engine(
        input_file=mapped_validated_results, engine="unify_csv"
    )

    filtered_validated_results = uc.execute_misc_engine(
        input_file=unified_validated_results,
        engine="filter_csv",
    )

    return


if __name__ == "__main__":
    main(input_raw_file=sys.argv[1], database=sys.argv[2], enzyme=sys.argv[3])
```

Processing of mass differences using PTM-Shepherd

ptmshepherd_mass_difference_processing.**main**(*folder=None*, *sanitized_search_results=None*)

Downstream processing of mass differences from open modification search results using PTM-Shepherd. A folder containing all relevant mzML files and a sanitized (one PSM per spectrum) Ursgal result file (containing open modification search results) are required.

usage: ./ptmshepherd_mass_difference_processing.py <folder_with_mzML> <sanitized_result_file>

```
#!/usr/bin/env python3
import os
import ursgal
import sys
import glob
from collections import defaultdict as ddict
import csv


def main(folder=None, sanitized_search_results=None):
    """
    Downstream processing of mass differences from open modification search results
    using PTM-Shepherd.
    A folder containing all relevant mzML files and a sanitized (one PSM per spectrum)
    Ursgal result file (containing open modification search results) are required.

    usage:
        ./ptmshepherd_mass_difference_processing.py <folder_with_mzML> <sanitized_
↪result_file>

    """
    mzml_files = []
    for mzml in glob.glob(os.path.join(folder, "*.mzML")):
        mzml_files.append(mzml)
```

(continues on next page)

(continued from previous page)

```
    params = {
        "use_pyqms_for_mz_calculation": True,
        "cpus": 8,
        "precursor_mass_tolerance_unit": "ppm",
        "precursor_mass_tolerance_plus": 5,
        "precursor_mass_tolerance_minus": 5,
        "frag_mass_tolerance_unit": "ppm",
        "frag_mass_tolerance": 20,
        "modifications": [
            "C,opt,any,Carbamidomethyl",
        ],
        "-xmx": "12G",
        "psm_defining_colnames": [
            "Spectrum Title",
            "Sequence",
            # 'Modifications',
            # 'Mass Difference',
            # 'Charge',
            # 'Is decoy',
        ],
        "mzml_input_files": mzml_files,
        "validation_score_field": "combined PEP",
        "bigger_scores_better": False,
    }

    uc = ursgal.UController(params=params, profile="QExactive+", verbose=False)

    ptmshepherd_results = uc.validate(
        input_file=sanitized_search_results,
        engine="ptmshepherd_0_3_5",
        # force=True,
    )


if __name__ == "__main__":
    main(
        folder=sys.argv[1],
        sanitized_search_results=sys.argv[2],
    )
```

7.1.4 Version Comparison Scripts

X!Tandem version comparison

xtandem_version_comparison.**main**()

Executes a search with 5 versions of X!Tandem on an example file from the data from Barth et al. 2014.

usage: ./xtandem_version_comparison.py

This is a simple example file to show the straightforward comparison of different program versions of X!Tandem

Creates a Venn diagram with the peptides obtained by the different versions.

Note: At the moment in total 6 XTandem versions are incorporated in Ursgal.

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

import ursgal
import os


def main():
    """
    Executes a search with 5 versions of X!Tandem on an example file from the
    data from Barth et al. 2014.

    usage:
        ./xtandem_version_comparison.py


    This is a simple example file to show the straightforward comparison of
    different program versions of X!Tandem

    Creates a Venn diagram with the peptides obtained by the different versions.

    Note:
        At the moment in total 6 XTandem versions are incorporated in Ursgal.

    """
    engine_list = [
        # 'xtandem_cyclone',
        "xtandem_jackhammer",
        "xtandem_sledgehammer",
        "xtandem_piledriver",
        "xtandem_vengeance",
        "xtandem_alanine",
    ]

    params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "modifications": [],
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_include_ext": [
            "JB_FASP_pH8_2-3_28122012.mzML",
        ],
        "ftp_output_folder": os.path.join(
            os.pardir, "example_data", "xtandem_version_comparison"
        ),
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
    }

    if os.path.exists(params["ftp_output_folder"]) is False:
```

(continues on next page)

(continued from previous page)

```
        os.mkdir(params["ftp_output_folder"])

    uc = ursgal.UController(profile="LTQ XL low res", params=params)
    mzML_file = os.path.join(params["ftp_output_folder"], params["ftp_include_ext
↪"][0])
    if os.path.exists(mzML_file) is False:
        uc.fetch_file(engine="get_ftp_files_1_0_0")
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    filtered_files_list = []
    for engine in engine_list:
        unified_result_file = uc.search(
            input_file=mzML_file,
            engine=engine,
        )

        validated_file = uc.validate(
            input_file=unified_result_file,
            engine="percolator_2_08",
        )

        filtered_file = uc.execute_misc_engine(
            input_file=validated_file,
            engine="filter_csv_1_0_0",
        )

        filtered_files_list.append(filtered_file)

    uc.visualize(
        input_files=filtered_files_list,
        engine="venndiagram_1_1_0",
    )
    return


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

MSGF+ version comparison

msgf_version_comparison.**main**()

Executes a search with a current maximum of 3 versions of MSGF+ on an example file from the data from Barth et al. (2014)

usage: ./msgf_version_comparison.py

This is a simple example file to show the straightforward comparison of different program versions of MSGF+

Creates a Venn diagram with the peptides obtained by the different versions.

An example plot can be found in the online documentation.

Note: Uses the new MS-GF+ C# mzid converter if available

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

import ursgal
import os


def main():
    """
    Executes a search with a current maximum of 3 versions of MSGF+ on an
    example file from the data from Barth et al. (2014)

    usage:
        ./msgf_version_comparison.py


    This is a simple example file to show the straightforward comparison of
    different program versions of MSGF+

    Creates a Venn diagram with the peptides obtained by the different
    versions.

    An example plot can be found in the online documentation.

    Note:
        Uses the new MS-GF+ C# mzid converter if available

    """

    params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]],
        "ftp_url": "ftp.peptideatlas.org",
        "ftp_login": "PASS00269",
        "ftp_password": "FI4645a",
        "ftp_include_ext": [
            "JB_FASP_pH8_2-3_28122012.mzML",
        ],
        "ftp_output_folder": os.path.join(
            os.pardir, "example_data", "msgf_version_comparison"
        ),
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
        "remove_temporary_files": False,
    }

    if os.path.exists(params["ftp_output_folder"]) is False:
        os.mkdir(params["ftp_output_folder"])

    uc = ursgal.UController(profile="LTQ XL low res", params=params)

    engine_list = [
```

(continues on next page)

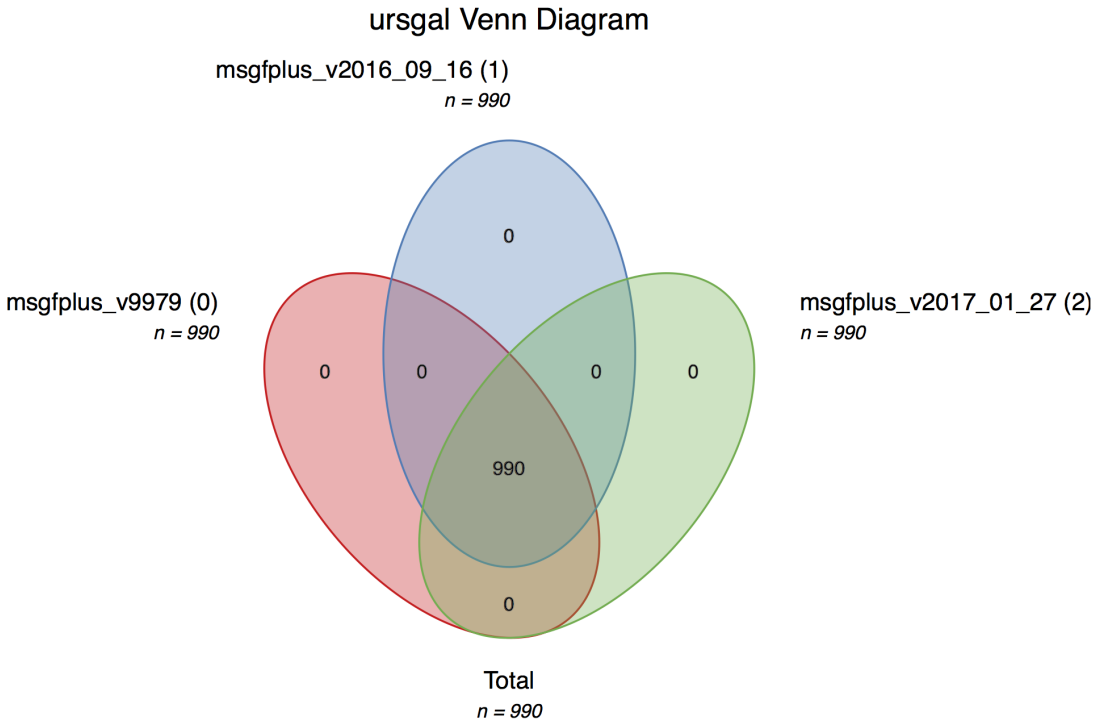
(continued from previous page)

```
        "msgfplus_v9979",
        "msgfplus_v2016_09_16",
    ]
    if "msgfplus_v2017_01_27" in uc.unodes.keys():
        if uc.unodes["msgfplus_v2017_01_27"]["available"]:
            engine_list.append("msgfplus_v2017_01_27")
    if "msgfplus_v2018_01_30" in uc.unodes.keys():
        if uc.unodes["msgfplus_v2018_01_30"]["available"]:
            engine_list.append("msgfplus_v2018_01_30")

    if "msgfplus_C_mzid2csv_v2017_07_04" in uc.unodes.keys():
        if uc.unodes["msgfplus_C_mzid2csv_v2017_07_04"]["available"]:
            uc.params[
                "msgfplus_mzid_converter_version"
            ] = "msgfplus_C_mzid2csv_v2017_07_04"

    mzML_file = os.path.join(params["ftp_output_folder"], params["ftp_include_ext
↪"][0])
    if os.path.exists(mzML_file) is False:
        uc.fetch_file(engine="get_ftp_files_1_0_0")
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    filtered_files_list = []
    for engine in engine_list:
        unified_result_file = uc.search(
            input_file=mzML_file,
            engine=engine,
        )

        validated_file = uc.validate(
            input_file=unified_result_file,
            engine="percolator_2_08",
        )

        filtered_file = uc.execute_misc_engine(
            input_file=validated_file,
            engine="filter_csv_1_0_0",
        )

        filtered_files_list.append(filtered_file)

    uc.visualize(
        input_files=filtered_files_list,
        engine="venndiagram_1_1_0",
    )
    return


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

Bruderer et al. (2015) X!Tandem comparison incl. machine ppm offset

xtandem_version_comparison_qexactive.**main**(*folder*)

Executes a search with 5 versions of X!Tandem on an example file from the data from Bruderer et al. 2015.


ursgal Venn Diagram
msgfplus_v2016_09_16 (1)
n = 990
msgfplus_v9979 (0)
n = 990
msgfplus_v2017_01_27 (2)
n = 990
0
0
0
0
0
990
0
Total
n = 990

usage: ./xtandem_version_comparison.py <folder containing B_D140314_SGSDSsample1_R01_MSG_T0.mzML>

This is a simple example file to show the straightforward comparison of different program versions of X!Tandem, similar to the example script 'xtandem_version_comparison', but analyzing high resolution data which can be better handled by version newer than Jackhammer. One gains approximately 10 percent more peptides with newer versions of X!Tandem.

Creates a Venn diagram with the peptides obtained by the different versions.

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

import ursgal
import os
import sys


def main(folder):
    """
    Executes a search with 5 versions of X!Tandem on an example file from the
    data from Bruderer et al. 2015.

    usage:
        ./xtandem_version_comparison.py <folder containing B_D140314_SGSDSsample1_R01_
↪MSG_T0.mzML>


    This is a simple example file to show the straightforward comparison of
    different program versions of X!Tandem, similar to the example script
    'xtandem_version_comparison', but analyzing high resolution data
    which can be better handled by version newer than Jackhammer. One gains
    approximately 10 percent more peptides with newer versions of X!Tandem.

    Creates a Venn diagram with the peptides obtained by the different versions.


    """

    required_example_file = "B_D140314_SGSDSsample1_R01_MSG_T0.mzML"

    if os.path.exists(os.path.join(folder, required_example_file)) is False:
        print(
            """
            Your specified folder does not contain the required example file:
            {0}
            The RAW data from peptideatlas.org (PASS00589, password: WF6554orn)
            will be downloaded.
            Please convert to mzML after the download has finished and run this
            script again.
            """.format(
                required_example_file
            )
        )

        ftp_get_params = {
            "ftp_url": "ftp.peptideatlas.org",
            "ftp_login": "PASS00589",
            "ftp_password": "WF6554orn",
```

(continues on next page)

(continued from previous page)

```
            "ftp_include_ext": [required_example_file.replace(".mzML", ".raw")],
            "ftp_output_folder": folder,
        }
        uc = ursgal.UController(params=ftp_get_params)
        uc.fetch_file(engine="get_ftp_files_1_0_0")
        sys.exit(1)

    engine_list = [
        "xtandem_cyclone",
        "xtandem_jackhammer",
        "xtandem_sledgehammer",
        "xtandem_piledriver",
        "xtandem_vengeance",
    ]

    params = {
        "database": os.path.join(
            os.pardir, "example_data", "hs_201303_qs_sip_target_decoy.fasta"
        ),
        "modifications": ["C,fix,any,Carbamidomethyl"],
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]],
        "http_url": "http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/hs_
↪201303_qs_sip_target_decoy.fasta",
        "http_output_folder": os.path.join(os.pardir, "example_data"),
        "machine_offset_in_ppm": -5e-6,
    }

    uc = ursgal.UController(profile="QExactive+", params=params)

    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    mzML_file = os.path.join(folder, required_example_file)

    filtered_files_list = []
    for engine in engine_list:

        unified_result_file = uc.search(
            input_file=mzML_file,
            engine=engine,
            force=False,
        )

        validated_file = uc.validate(
            input_file=unified_result_file,
            engine="percolator_2_08",
        )

        filtered_file = uc.execute_misc_engine(
            input_file=validated_file,
            engine="filter_csv_1_0_0",
        )

        filtered_files_list.append(filtered_file)

    uc.visualize(
        input_files=filtered_files_list,
```

(continues on next page)

(continued from previous page)

```
        engine="venndiagram_1_1_0",
    )
    return


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(main.__doc__)
        sys.exit(1)
    main(sys.argv[1])
```

7.1.5 Parameter Optimization Scripts

BSA machine ppm offset example

bsa_ppm_offset_test.**main**()

Example script to do a simple machine ppm offset parameter sweep. The m/z values in the example mgf file are stepwise changed and the in the final output the total peptides are counted.

usage: ./bsa_ppm_offset_test.py

Note: As expected, if the offset becomes to big no peptides can be found anymore.

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

import ursgal
import glob
import csv
from collections import defaultdict as ddict
import os
import shutil


def main():
    """
    Example script to do a simple machine ppm offset parameter sweep.
    The m/z values in the example mgf file are stepwise changed and the in the
    final output the total peptides are counted.

    usage:
        ./bsa_ppm_offset_test.py

    Note:
        As expected, if the offset becomes to big no peptides can be found anymore.

    """
    ppm_offsets = [
        (-10, "-10_ppm_offset"),
        (-9, "-9_ppm_offset"),
        (-8, "-8_ppm_offset"),
        (-7, "-7_ppm_offset"),
        (-6, "-6_ppm_offset"),
```

(continues on next page)

(continued from previous page)

```
        (-5, "-5_ppm_offset"),
        (-4, "-4_ppm_offset"),
        (-3, "-3_ppm_offset"),
        (-2, "-2_ppm_offset"),
        (-1, "-1_ppm_offset"),
        (None, "0_ppm_offset"),
        (1, "1_ppm_offset"),
        (2, "2_ppm_offset"),
        (3, "3_ppm_offset"),
        (4, "4_ppm_offset"),
        (5, "5_ppm_offset"),
        (6, "6_ppm_offset"),
        (7, "7_ppm_offset"),
        (8, "8_ppm_offset"),
        (9, "9_ppm_offset"),
        (10, "10_ppm_offset"),
    ]

    engine_list = ["xtandem_vengeance"]

    R = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "database": os.path.join(os.pardir, "example_data", "BSA.fasta"),
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
        },
    )

    mzML_file = os.path.join(
        os.pardir, "example_data", "BSA_machine_ppm_offset_example", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:
        R.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        R.params["http_output_folder"] = os.path.dirname(mzML_file)
        R.fetch_file(engine="get_http_files_1_0_0")
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)

    for engine in engine_list:
        for (ppm_offset, prefix) in ppm_offsets:

            R.params["machine_offset_in_ppm"] = ppm_offset
            R.params["prefix"] = prefix

            unified_search_result_file = R.search(
                input_file=mzML_file,
                engine=engine,
                force=False,
```

(continues on next page)

(continued from previous page)

```
            )

    collector = ddict(set)
    for csv_path in glob.glob("{0}/*/*unified.csv".format(os.path.dirname(mzML_
↪file))):
        for line_dict in csv.DictReader(open(csv_path, "r")):
            collector[csv_path].add(line_dict["Sequence"])
    for csv_path, peptide_set in sorted(collector.items()):
        file_name = os.path.basename(csv_path)
        offset = file_name.split("_")[0]
        print(
            "Search with {0: >3} ppm offset found {1: >2} peptides".format(
                offset, len(peptide_set)
            )
        )

    return


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

Precursor mass tolerance example

bsa_precursor_mass_tolerance_example.**main**()

Example script to do a precursor mass tolerance parameter sweep.

usage: ./bsa_precursor_mass_tolerance_example.py

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

import ursgal
import glob
import csv
from collections import defaultdict as ddict
import os
import shutil


def main():
    """
    Example script to do a precursor mass tolerance parameter sweep.

    usage:
        ./bsa_precursor_mass_tolerance_example.py


    """
    precursor_mass_tolerance_list = [
        1,
        2,
        3,
        4,
        5,
```

(continues on next page)

(continued from previous page)

```
        6,
        7,
        8,
        9,
        10,
    ]

    engine_list = ["xtandem_vengeance"]

    R = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "database": os.path.join(os.pardir, "example_data", "BSA.fasta"),
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
        },
    )

    mzML_file = os.path.join(
        os.pardir, "example_data", "BSA_precursor_mass_tolerance_example", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:

        R.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"

        R.params["http_output_folder"] = os.path.dirname(mzML_file)
        R.fetch_file(engine="get_http_files_1_0_0")
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)
    # Convert mzML to MGF outside the loop, so this step is not repeated in
    # the loop
    mgf_file = R.convert(input_file=mzML_file, engine="mzml2mgf_1_0_0")

    for engine in engine_list:
        for precursor_mass_tolerance in precursor_mass_tolerance_list:

            R.params["precursor_mass_tolerance_minus"] = precursor_mass_tolerance
            R.params["precursor_mass_tolerance_plus"] = precursor_mass_tolerance

            R.params["prefix"] = "{0}_precursor_mass_tolerance_".format(
                precursor_mass_tolerance
            )

            unified_search_result_file = R.search(
                input_file=mgf_file,
                engine=engine,
                force=False,
            )
```

(continues on next page)

(continued from previous page)

```
    collector = ddict(set)
    for csv_path in glob.glob("{0}/*/*unified.csv".format(os.path.dirname(mzML_
↪file))):
        for line_dict in csv.DictReader(open(csv_path, "r")):
            collector[csv_path].add(line_dict["Sequence"])
    for csv_path, peptide_set in sorted(collector.items()):
        file_name = os.path.basename(csv_path)
        tolerance = file_name.split("_")[0]
        print(
            "Search with {0: >2} ppm precursor mass tolerance found {1: >2} peptides".
↪format(
                tolerance, len(peptide_set)
            )
        )

    return


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

Fragment mass tolerance example

bsa_fragment_mass_tolerance_example.**main**()

Example script to do a fragment mass tolerance parameter sweep.

usage: ./bsa_fragment_mass_tolerance_example.py

If the fragment mass tolerance becomes to small, ver few peptides are found. With this small sweep the actual min accuracy of a mass spectrometer can be estimated.

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

import ursgal
import glob
import csv
from collections import defaultdict as ddict
import os
import shutil


def main():
    """
    Example script to do a fragment mass tolerance parameter sweep.

    usage:
        ./bsa_fragment_mass_tolerance_example.py

    If the fragment mass tolerance becomes to small, ver few peptides are found.
    With this small sweep the actual min accuracy of a mass spectrometer can
    be estimated.

    """
    fragment_mass_tolerance_list = [
        0.02,
```

(continues on next page)

(continued from previous page)

```
        0.04,
        0.06,
        0.08,
        0.1,
        0.2,
        0.3,
        0.4,
        0.5,
    ]

    engine_list = ["xtandem_vengeance"]

    R = ursgal.UController(
        profile="LTQ XL low res",
        params={
            "database": os.path.join(os.pardir, "example_data", "BSA.fasta"),
            "modifications": [
                "M,opt,any,Oxidation",  # Met oxidation
                "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
                "*,opt,Prot-N-term,Acetyl",  # N-Acteylation
            ],
        },
    )

    mzML_file = os.path.join(
        os.pardir, "example_data", "BSA_fragment_mass_tolerance_example", "BSA1.mzML"
    )
    if os.path.exists(mzML_file) is False:
        R.params[
            "http_url"
        ] = "http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/OpenMS/
↪examples/BSA/BSA1.mzML?format=raw"
        R.params["http_output_folder"] = os.path.dirname(mzML_file)
        R.fetch_file(engine="get_http_files_1_0_0")
        try:
            shutil.move("{0}?format=raw".format(mzML_file), mzML_file)
        except:
            shutil.move("{0}format=raw".format(mzML_file), mzML_file)

    # Convert mzML to MGF outside the loop, so this step is not repeated in
    # the loop
    mgf_file = R.convert(
        input_file=mzML_file,
        engine="mzml2mgf_1_0_0",
    )

    for engine in engine_list:
        for fragment_mass_tolerance in fragment_mass_tolerance_list:

            R.params["frag_mass_tolerance"] = fragment_mass_tolerance

            R.params["prefix"] = "{0}_fragment_mass_tolerance_".format(
                fragment_mass_tolerance
            )

            unified_search_result_file = R.search(
                input_file=mgf_file,
```

(continues on next page)

(continued from previous page)

```
                engine=engine,
                force=False,
            )

    collector = ddict(set)
    for csv_path in glob.glob("{0}/*/*unified.csv".format(os.path.dirname(mzML_
↪file))):
        for line_dict in csv.DictReader(open(csv_path, "r")):
            collector[csv_path].add(line_dict["Sequence"])
    for csv_path, peptide_set in sorted(collector.items()):
        file_name = os.path.basename(csv_path)
        tolerance = file_name.split("_")[0]
        print(
            "Search with {0: <4} Da fragment mass tolerance found {1: >2} peptides".
↪format(
                tolerance, len(peptide_set)
            )
        )
    return


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

Bruderer et al. (2015) machine offset sweep (data for figure 2)

machine_offset_bruderer_sweep.**search**(*input_folder=None*)

Does the parameter sweep on every tenth MS2 spectrum of the data from Bruderer et al. (2015) und X!Tandem Sledgehammer.

Note: Please download the .RAW data for the DDA dataset from peptideatlas.org (PASS00589, password: WF6554orn) and convert to mzML. Then the script can be executed with the folder with the mzML files as the first argument.

Warning: This script (if the sweep ranges are not changed) will perform 10080 searches which will produce approximately 100 GB output (inclusive mzML files)

usage:

./machine_offset_bruderer_sweep.py <folder_with_bruderer_data>

Sweeps over:

- machine_offset_in_ppm from -20 to +20 ppm offset
- precursor mass tolerance from 1 to 5 ppm
- fragment mass tolerance from -2.5 to 20 ppm

The search can be very time consuming (depending on your machine/cluster), therefor the analyze step can be performed separately by calling analyze() instead of search() when one has already performed the searches and wants to analyze the results.

machine_offset_bruderer_sweep.**analyze**(*folder*)

Parses the result files form search and write a result .csv file which contains the data to plot figure 2.

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

import ursgal
import glob
import csv
import os
from collections import defaultdict as ddict
import sys
import re


MQ_OFFSET_TO_FILENAME = [
    (4.71, "B_D140314_SGSDSsample2_R01_MSG_T0.mzML"),
    (4.98, "B_D140314_SGSDSsample6_R01_MSG_T0.mzML"),
    (4.9, "B_D140314_SGSDSsample1_R01_MSG_T0.mzML"),
    (5.41, "B_D140314_SGSDSsample4_R01_MSG_T0.mzML"),
    (5.78, "B_D140314_SGSDSsample5_R01_MSG_T0.mzML"),
    (6.01, "B_D140314_SGSDSsample8_R01_MSG_T0.mzML"),
    (6.22, "B_D140314_SGSDSsample7_R01_MSG_T0.mzML"),
    (6.83, "B_D140314_SGSDSsample3_R01_MSG_T0.mzML"),
    (7.61, "B_D140314_SGSDSsample4_R02_MSG_T0.mzML"),
    (7.59, "B_D140314_SGSDSsample8_R02_MSG_T0.mzML"),
    (7.93, "B_D140314_SGSDSsample6_R02_MSG_T0.mzML"),
    (7.91, "B_D140314_SGSDSsample1_R02_MSG_T0.mzML"),
    (8.23, "B_D140314_SGSDSsample3_R02_MSG_T0.mzML"),
    (8.33, "B_D140314_SGSDSsample7_R02_MSG_T0.mzML"),
    (9.2, "B_D140314_SGSDSsample5_R02_MSG_T0.mzML"),
    (9.4, "B_D140314_SGSDSsample2_R02_MSG_T0.mzML"),
    (9.79, "B_D140314_SGSDSsample1_R03_MSG_T0.mzML"),
    (10.01, "B_D140314_SGSDSsample3_R03_MSG_T0.mzML"),
    (10.03, "B_D140314_SGSDSsample7_R03_MSG_T0.mzML"),
    (10.58, "B_D140314_SGSDSsample2_R03_MSG_T0.mzML"),
    (11.1, "B_D140314_SGSDSsample4_R03_MSG_T0.mzML"),
    (11.21, "B_D140314_SGSDSsample5_R03_MSG_T0.mzML"),
    (11.45, "B_D140314_SGSDSsample6_R03_MSG_T0.mzML"),
    (12.19, "B_D140314_SGSDSsample8_R03_MSG_T0.mzML"),
]

GENERAL_PARAMS = {
    "database": os.path.join(
        os.pardir, "example_data", "hs_201303_qs_sip_target_decoy.fasta"
    ),
    "modifications": [
        "C,fix,any,Carbamidomethyl",  # Carbamidomethylation
    ],
    "scan_skip_modulo_step": 10,
    "http_url": "http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/hs_201303_qs_
↪sip_target_decoy.fasta",
    "http_output_folder": os.path.join(os.pardir, "example_data"),
}


def search(input_folder=None):
    """
```

(continues on next page)

(continued from previous page)

```
    Does the parameter sweep on every tenth MS2 spectrum of the data from
    Bruderer et al. (2015) und X!Tandem Sledgehammer.

    Note:

        Please download the .RAW data for the DDA dataset from peptideatlas.org
        (PASS00589, password: WF6554orn) and convert to mzML.
        Then the script can be executed with the folder with the mzML files as
        the first argument.

    Warning:

        This script (if the sweep ranges are not changed) will perform 10080
        searches which will produce approximately 100 GB output (inclusive mzML
        files)

    usage:

        ./machine_offset_bruderer_sweep.py <folder_with_bruderer_data>

    Sweeps over:

        * machine_offset_in_ppm from -20 to +20 ppm offset
        * precursor mass tolerance from 1 to 5 ppm
        * fragment mass tolerance from -2.5 to 20 ppm

    The search can be very time consuming (depending on your machine/cluster),
    therefor the analyze step can be performed separately by calling analyze()
    instead of search() when one has already performed the searches and wants
    to analyze the results.
    """

    file_2_target_offset = {}
    for MQ_offset, file_name in MQ_OFFSET_TO_FILENAME:
        file_2_target_offset[file_name] = {"offsets": []}
        for n in range(-20, 20, 2):
            file_2_target_offset[file_name]["offsets"].append(n)

    engine_list = ["xtandem_sledgehammer"]

    frag_ion_tolerance_list = [2.5, 5, 10, 20]

    precursor_ion_tolerance_list = [1, 2, 3, 4, 5]

    R = ursgal.UController(profile="QExactive+", params=GENERAL_PARAMS)

    if os.path.exists(R.params["database"]) is False:
        R.fetch_file(engine="get_http_files_1_0_0")

    prefix_format_string = "_pit_{1}_fit_{2}"
    for mzML_path in glob.glob(os.path.join(input_folder, "*.mzML")):

        mzML_basename = os.path.basename(mzML_path)
        if mzML_basename not in file_2_target_offset.keys():
            continue
        for ppm_offset in file_2_target_offset[mzML_basename]["offsets"]:
            R.params["machine_offset_in_ppm"] = ppm_offset
```

(continues on next page)

(continued from previous page)

```
            R.params["prefix"] = "ppm_offset_{0}".format(int(ppm_offset))
            mgf_file = R.convert(input_file=mzML_path, engine="mzml2mgf_1_0_0")
            for engine in engine_list:
                for precursor_ion_tolerane in precursor_ion_tolerance_list:
                    for frag_ion_tolerance in frag_ion_tolerance_list:

                        new_prefix = prefix_format_string.format(
                            precursor_ion_tolerane, frag_ion_tolerance
                        )
                        R.params[
                            "precursor_mass_tolerance_minus"
                        ] = precursor_ion_tolerane
                        R.params[
                            "precursor_mass_tolerance_plus"
                        ] = precursor_ion_tolerane
                        R.params["frag_mass_tolerance"] = frag_ion_tolerance
                        R.params["prefix"] = new_prefix

                        unified_search_result_file = R.search(
                            input_file=mzML_path,
                            engine=engine,
                            force=False,
                        )

    return


def analyze(folder):
    """

    Parses the result files form search and write a result .csv file which
    contains the data to plot figure 2.

    """

    R = ursgal.UController(profile="QExactive+", params=GENERAL_PARAMS)
    csv_collector = {}
    ve = "qvality_2_02"

    sample_regex_pattern = "sample\d_R0\d"

    sample_2_x_pos_and_mq_offset = {}

    sample_offset_combos = []

    all_tested_offsets = [str(n) for n in range(-20, 21, 2)]

    for pos, (mq_ppm_off, mzML_file) in enumerate(MQ_OFFSET_TO_FILENAME):
        _sample = re.search(sample_regex_pattern, mzML_file).group()
        sample_2_x_pos_and_mq_offset[_sample] = (pos, mq_ppm_off)
        for theo_offset in all_tested_offsets:
            sample_offset_combos.append((_sample, theo_offset))

    for csv_path in glob.glob(os.path.join("{0}".format(folder), "*", "*_unified.csv
↪")):
        dirname = os.path.dirname(csv_path)
        sample = re.search(sample_regex_pattern, csv_path).group()
```

(continues on next page)

(continued from previous page)

```
        splitted_basename = os.path.basename(csv_path).split("_")
        offset = splitted_basename[2]
        precursor_ion_tolerance = splitted_basename[4]
        frag_ion_tolerance = splitted_basename[6]
        prefix = "_".join(splitted_basename[:7])

        R.params["machine_offset_in_ppm"] = offset
        R.params["precursor_mass_tolerance_minus"] = precursor_ion_tolerance
        R.params["precursor_mass_tolerance_plus"] = precursor_ion_tolerance
        R.params["frag_mass_tolerance"] = frag_ion_tolerance
        R.params["prefix"] = prefix

        validated_path = csv_path.replace(
            "_unified.csv", "_{0}_validated.csv".format(ve)
        )
        if os.path.exists(validated_path):
            csv_path = validated_path
        else:
            try:
                csv_path = R.validate(input_file=csv_path, engine=ve)
            except:
                continue

        pit_fit = (precursor_ion_tolerance, frag_ion_tolerance)

        if pit_fit not in csv_collector.keys():
            csv_collector[pit_fit] = ddict(set)

        csv_key = (sample, offset)

        print("Reading file: {0}".format(csv_path))
        for line_dict in csv.DictReader(open(csv_path, "r")):
            if line_dict["Is decoy"] == "true":
                continue
            if float(line_dict["PEP"]) <= 0.01:
                csv_collector[pit_fit][csv_key].add(
                    "{0}{1}".format(line_dict["Sequence"], line_dict["Modifications"])
                )

    fieldnames = ["Sample", "pos", "MQ_offset", "tested_ppm_offset", "peptide_count"]

    outfile_name_format_string = "bruderer_data_ppm_sweep_precursor_mass_tolerance_{0}
↪_fragment_mass_tolerance_{1}.csv"

    for pit_fit in csv_collector.keys():
        with open(outfile_name_format_string.format(*pit_fit), "w") as io:
            csv_writer = csv.DictWriter(io, fieldnames)
            csv_writer.writeheader()

            # write missing values
            for sample_offset in sample_offset_combos:
                sample, ppm_offset = sample_offset
                if sample_offset not in csv_collector[pit_fit].keys():
                    dict_2_write = {
                        "Sample": sample,
                        "pos": sample_2_x_pos_and_mq_offset[sample][0],
                        "MQ_offset": "",
```

(continues on next page)

(continued from previous page)

```
                        "tested_ppm_offset": ppm_offset,
                        "peptide_count": 0,
                    }
                    csv_writer.writerow(dict_2_write)

            for (sample, ppm_offset), peptide_set in csv_collector[pit_fit].items():
                dict_2_write = {
                    "Sample": sample,
                    "pos": sample_2_x_pos_and_mq_offset[sample][0],
                    "MQ_offset": sample_2_x_pos_and_mq_offset[sample][1] * -1,
                    "tested_ppm_offset": ppm_offset,
                    "peptide_count": len(peptide_set),
                }
                csv_writer.writerow(dict_2_write)
    return


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(search.__doc__)
        sys.exit(1)
    search(sys.argv[1])
    analyze(sys.argv[1])
```

7.1.6 Filter CSV Examples

Filter for modifications

filter_csv_for_mods_example.**main**()

Examples script for filtering unified results for modification containing entries

usage: ./filter_csv_for_mods_example.py

Will produce a file with only entries which contain Carbamidomethyl as a modification.

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

import ursgal
import os


def main():
    """
    Examples script for filtering unified results for modification containing
    entries

    usage:
        ./filter_csv_for_mods_example.py


    Will produce a file with only entries which contain Carbamidomethyl as a
    modification.
    """
    params = {
```

(continues on next page)

(continued from previous page)

```
        "csv_filter_rules": [
            ["Modifications", "contains", "Carbamidomethyl"],
        ],
        "write_unfiltered_results": False,
    }

    csv_file_to_filter = os.path.join(
        os.pardir,
        "example_data",
        "misc",
        "filter_csv_for_mods_example_omssa_2_1_9_pmap_unified.csv",
    )
    uc = ursgal.UController(params=params)

    filtered_csv = uc.execute_misc_engine(
        input_file=csv_file_to_filter,
        engine="filter_csv_1_0_0",
    )


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

Filter validated results

`filter_csv_validation_example.main()`

Examples script for filtering validated results for a PEP <= 0.01 and remove all decoys.

usage: ./filter_csv_validation_example.py

Will produce a file with only target sequences with a posterior error probability of lower or equal to 1 percent

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

import ursgal
import os


def main():
    """
    Examples script for filtering validated results for a PEP <= 0.01 and
    remove all decoys.

    usage:
        ./filter_csv_validation_example.py


    Will produce a file with only target sequences with a posterior error
    probability of lower or equal to 1 percent
    """
    params = {
        "csv_filter_rules": [["PEP", "lte", 0.01], ["Is decoy", "equals", "false"]]
    }

    csv_file_to_filter = os.path.join(
```

(continues on next page)

(continued from previous page)

```
        os.pardir,
        "example_data",
        "misc",
        "filter_csv_for_mods_example_omssa_2_1_9_pmap_unified_percolator_2_08_
↪validated.csv",
    )
    uc = ursgal.UController(params=params)

    filtered_csv = uc.execute_misc_engine(
        input_file=csv_file_to_filter,
        engine="filter_csv_1_0_0",
    )


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

7.1.7 SugarPy Example Scripts

SugarPy complete workflow

sugarpy_example_workflow.**main**(*folder=None*, *target_decoy_database=None*)

Complete example workflow for the identification of intact glycopeptides from MS runs employing IS-CID.

usage: ./do_it_all_folder_wide.py <mzML_folder> <target_decoy_database>

```
#!/usr/bin/env python3.4
# encoding: utf-8
import ursgal
import sys
import glob
import os
import shutil


def main(folder=None, target_decoy_database=None):
    '''
    Complete example workflow for the identification of intact glycopeptides
    from MS runs employing IS-CID.

    usage:
        ./do_it_all_folder_wide.py <mzML_folder> <target_decoy_database>

    '''
    # define folder with mzML_files as sys.argv[1]
    mzML_files = []
    for mzml in glob.glob(os.path.join('{0}'.format(folder), '*.mzML')):
        mzML_files.append(mzml)

    mass_spectrometer = 'QExactive+'

    # Define search engines for protein database search
    search_engines = [
        'xtandem_vengeance',
        'msgfplus_v2019_07_03',
```

(continues on next page)

(continued from previous page)

```
    'msfragger_2_3',
]

# Define validation engine for protein database search
validation_engine = 'percolator_3_4_0'

# Modifications that should be included in the search.
# Glycan modifications will be added later.
all_mods = [
    'C,fix,any,Carbamidomethyl',
    'M,opt,any,Oxidation',
    '*,opt,Prot-N-term,Acetyl',
]

# Initializing the Ursgal UController class with
# our specified modifications and mass spectrometer
params = {
    'cpus':8,
    'database'      : target_decoy_database,
    'modifications' : all_mods,
    'csv_filter_rules' : [
        ['Is decoy', 'equals', 'false'],
        ['PEP', 'lte', 0.01],
        ['Conflicting uparam', 'contains_not', 'enzyme'],
    ],
    'peptide_mapper_class_version': 'UPeptideMapper_v4',
    'use_pyqms_for_mz_calculation': True,
    '-xmx'     : '16g',
    'upper_mz_limit': 3000,
    'lower_mz_limit': 500,
    'precursor_mass_tolerance_plus': 5,
    'precursor_mass_tolerance_minus': 5,
    'frag_mass_tolerance': 20,
}

uc = ursgal.UController(
    profile=mass_spectrometer,
    params=params
)

# complete workflow
result_files_unfilered = []
glyco_result_files = []
pot_glyco_result_files = []
sugarpy_result_files = []
sugarpy_curated_files = []
for spec_file in mzML_files:
    mgf_file = uc.convert(
        input_file=spec_file,
        engine='mzml2mgf_2_0_0'
    )
    results_one_file = []
    for search_engine in search_engines:
        # search MS2 specs for peptides with HexNAc and HexNAc(2) modification
        for mod in [
            'N,opt,any,HexNAc',
            'N,opt,any,HexNAc(2)'
```

(continues on next page)

(continued from previous page)

```
            ]:
                uc.params['modifications'].append(mod)
                uc.params['prefix'] = mod.split(',')[3]
                search_result = uc.search_mgf(
                    input_file=mgf_file,
                    engine=search_engine
                )
                uc.params['prefix'] = ''
                converted_result = uc.convert(
                    input_file=search_result,
                    guess_engine=True,
                )
                mapped_results = uc.execute_misc_engine(
                    input_file=converted_result,
                    engine='upeptide_mapper'
                )
                unified_search_results = uc.execute_misc_engine(
                    input_file=mapped_results,
                    engine='unify_csv',
                )
                validated_csv = uc.validate(
                    input_file=unified_search_results,
                    engine=validation_engine,
                )
                filtered_validated_results = uc.execute_misc_engine(
                    input_file=validated_csv,
                    engine='filter_csv',
                )
                results_one_file.append(filtered_validated_results)
                uc.params['modifications'].remove(mod)

        uc.params['prefix'] = '{0}'.format(
            os.path.basename(spec_file).strip('.mzML'))

        all_results_one_file = uc.execute_misc_engine(
            input_file=results_one_file,
            engine='merge_csvs',
        )
        result_files_unfilered.append(all_results_one_file)

        # filter for glycopeptides
        uc.params.update({
            'prefix': 'Glyco_',
            'csv_filter_rules': [
                ['Modifications', 'contains', 'HexNAc'],
                ['Modifications', 'mod_at_glycosite', 'HexNAc'],
            ]
        })
        glyco_file = uc.execute_misc_engine(
            input_file=all_results_one_file,
            engine='filter_csv',
        )
        glyco_result_files.append(glyco_file)

        # sugarpy_run to match glycopeptide fragments in MS1 scans.
        # Modifiy those parameters according your MS specifications
        uc.params.update({
```

(continues on next page)

(continued from previous page)

```
            'prefix': '',
            'mzml_input_file': spec_file,
            'min_number_of_spectra': 2,
            'min_glycan_length': 3,
            'max_glycan_length': 20,
            'min_subtree_coverage': 0.65,
            'min_sugarpy_score': 1,
            'ms_level': 1,
            'rt_border_tolerance': 1,
            'precursor_max_charge': 5,
            'use_median_accuracy': 'peptide',
            'monosaccharide_compositions': {
                "dHex": 'C6H10O4',
                "Hex": 'C6H10O5',
                "HexNAc": 'C8H13NO5',
                "NeuAc": 'C11H17NO8',
                # "dHexNAc": 'C8H13NO4',
                # "HexA": 'C6H8O6',
                # "Me2Hex": 'C8H14O5',
                # "MeHex": 'C7H12O5',
                # "Pent": 'C5H8O4',
                # 'dHexN': 'C6H11O3N',
                # 'HexN': 'C6H11O4N',
                # 'MeHexA': 'C7H10O6',
            },
            'm_score_cutoff': 0.7,
            'mz_score_percentile': 0.7,
            'precursor_mass_tolerance_plus': 10,
            'precursor_mass_tolerance_minus': 10,
            'rel_intensity_range': 0.2,
            'rt_pickle_name': os.path.join(
                os.path.dirname(spec_file),
                '_ursgal_lookup.pkl'
            )
        })
        sugarpy_results = uc.execute_misc_engine(
            input_file=glyco_file,
            engine='sugarpy_run_1_0_0',
        )
        sugarpy_result_files.append(sugarpy_results)

        # sugarpy_plot to plot the results from the sugarpy_run
        # This will plot an elution profile for all identified glycopeptides
        uc.params.update({
            'sugarpy_results_pkl': sugarpy_results.replace('.csv', '.pkl'),
            'sugarpy_results_csv': sugarpy_results,
            'sugarpy_plot_types': [
                'plot_glycan_elution_profile',
            ],
            'plotly_layout': {
                'font' : {
                    'family':'Arial',
                    # 'size': 20,
                    'color' : 'rgb(0, 0, 0)',
                },
                'autosize':True,
                # 'width':1700,
```

(continues on next page)

(continued from previous page)

```
                # 'height':700,
                # 'margin':{
                #     'l':100,
                #     'r':80,
                #     't':100,
                #     'b':60,
                # },
                'xaxis':{
                    'type':'linear',
                    'color':'rgb(0, 0, 0)',
                    'title_font':{
                        'family':'Arial',
                        'size':20,
                        'color':'rgb(0, 0, 0)',
                    },
                    'autorange':True,
                    # 'range':[0,1000.0],
                    'tickmode':'auto',
                    'showexponent':'all',
                    'exponentformat':'B',
                    'ticklen':5,
                    'tickwidth':1,
                    'tickcolor':'rgb(0, 0, 0)',
                    'ticks':'outside',
                    'showline':True,
                    'linecolor':'rgb(0, 0, 0)',
                    'mirror':False,
                    'showgrid':False,
                },
                'yaxis':{
                    'type':'linear',
                    'color':'rgb(0, 0, 0)',
                    'title':'Intensity',
                    'title_font':{
                        'family':'Arial',
                        'size':20,
                        'color':'rgb(0, 0, 0)',
                    },
                    'autorange':True,
                    # 'range':[0.0,100.0],
                    'showexponent':'all',
                    'exponentformat':'B',
                    'ticklen':5,
                    'tickwidth':1,
                    'tickcolor':'rgb(0, 0, 0)',
                    'ticks':'outside',
                    'showline':True,
                    'linecolor':'rgb(0, 0, 0)',
                    'mirror':False,
                    'showgrid':False,
                },
                'legend':{
                    'orientation':'h',
                    'traceorder':'normal',
                },
                'showlegend':True,
                'paper_bgcolor':'rgba(0,0,0,0)',
```

(continues on next page)

(continued from previous page)

```
                'plot_bgcolor':'rgba(0,0,0,0)',
            },
        })
        sugarpy_plots = uc.execute_misc_engine(
            input_file=spec_file,
            engine='sugarpy_plot_1_0_0',
            # force = True,
        )

        # setting uparams back to original
        uc.params.update({
            'prefix': '',
            'ms_level': 2,
            'precursor_max_charge': 5,
            'csv_filter_rules': [
                ['Is decoy', 'equals', 'false'],
                ['PEP', 'lte', 0.01],
            ],
            'precursor_mass_tolerance_plus' : 5,
            'precursor_mass_tolerance_minus': 5,
            'rt_pickle_name': '_ursgal_lookup.pkl',
            'frag_mass_tolerance': 20,
            # 'rt_pickle_name': os.path.join(
            #     os.path.dirname(spec_file),
            #     '_ursgal_lookup.pkl'
            # )
        })

    # combine results from multiple mzML files
    results_all_files_unfiltered = uc.execute_misc_engine(
        input_file=result_files_unfilered,
        engine='merge_csvs',
    )
    glyco_results_all_files = uc.execute_misc_engine(
        input_file=glyco_result_files,
        engine='merge_csvs',
    )
    sugarpy_results_all_files = uc.execute_misc_engine(
        input_file=sugarpy_result_files,
        engine='merge_csvs',
    )

    # Extract the best matches from the raw SugarPy results.
    # This is used for the automatic filtering described in the manuscript.
    uc.params.update({
        'prefix': '',
        'ms_level': 1,
        'min_number_of_spectra': 2,
        'max_trees_per_spec': 1,
        'sugarpy_results_pkl': None,
        'sugarpy_results_csv': sugarpy_results_all_files,
        'sugarpy_plot_types': [
            'extract_best_matches',
        ],
        'rt_pickle_name': 'create_new_lookup',
    })
    sugarpy_extracted = uc.execute_misc_engine(
```

(continues on next page)

(continued from previous page)

```
        # the specific spec file is not important here,
        # this is just used to localize the scan2rt lookup
        input_file=spec_file,
        engine='sugarpy_plot_1_0_0',
        force = False,
    )

    # Filter extracted results for glycopeptides that were identified
    # in at least two replicates (i.e. two MS files)
    uc.params['csv_filter_rules'] = [
        ['Num Raw Files', 'gte', 2],
    ]

    extracted_filtered = uc.execute_misc_engine(
        input_file=sugarpy_results_all_files.replace('.csv', '_extracted.csv'),
        engine='filter_csv',
    )
if __name__ == '__main__':
    if len(sys.argv) < 3:
        print(main.__doc__)
        exit()
    main(
        folder=sys.argv[1],
        target_decoy_database=sys.argv[2],
    )
```

SugarPy plot annotated specs

sugarpy_plot_annotated_specs.**main**(*spec_file=None*, *sugarpy_results=None*, *sugarpy_results_pkl=None*)

This script can be used to plot any spectrum from SugarPy results. It woll plot all spectra listed in the input file (sugarpy_results), therefore it is recommended to provide a csv file with only a few, representative spectra for each glycopeptide. Besides the mzml file corresponding to the glycopeptide identifications, the sugarpy results csv file as well as the corresponding pkl file is required.

Usage: ./sugarpy_plot_annotated_specs.py <mzml.idx.gz> <sugarpy_results.csv> <sugarpy_results_pkl>

```
#!/usr/bin/env python3.4
# encoding: utf-8
import ursgal
import sys
import glob
import os
import shutil


def main(spec_file=None, sugarpy_results=None, sugarpy_results_pkl=None):
    '''
    This script can be used to plot any spectrum from SugarPy results.
    It woll plot all spectra listed in the input file (sugarpy_results),
    therefore it is recommended to provide a csv file with only a few,
    representative spectra for each glycopeptide.
    Besides the mzml file corresponding to the glycopeptide identifications,
    the sugarpy results csv file as well as the corresponding pkl file is required.
```

(continues on next page)

(continued from previous page)

```
    Usage:
        ./sugarpy_plot_annotated_specs.py <mzml.idx.gz> <sugarpy_results.csv>
↪<sugarpy_results_pkl>
    '''

    # initialize the UController
    uc = ursgal.UController()

    # sugarpy_plot to plot he results from the sugarpy_run
    uc.params.update({
        'prefix': 'Annotated_specs',
        'sugarpy_results_pkl': sugarpy_results_pkl,
        'sugarpy_results_csv': sugarpy_results,
        'sugarpy_plot_types': [
            'plot_annotated_spectra',
        ],
        'sugarpy_plot_peak_types': [
            'matched',
            'unmatched',
            'labels',
        ],
        # 'sugarpy_include_subtrees': 'individual_subtrees'
        'rt_pickle_name': os.path.join(
            os.path.dirname(spec_file),
            '_ursgal_lookup.pkl'
        ),
        'ms_level': 1,
        'plotly_layout': {
            'font' : {
                'family':'Arial',
                'size': 26,
                'color' : 'rgb(0, 0, 0)',
            },
            # 'autosize':True,
            'width':1750,
            'height':820,
            'margin':{
                'l':120,
                'r':50,
                't':50,
                'b':170,
            },
            'xaxis':{
                'type':'linear',
                'color':'rgb(0, 0, 0)',
                'title': 'm/z',
                'title_font':{
                    'family':'Arial',
                    'size':26,
                    'color':'rgb(0, 0, 0)',
                },
                'autorange':True,
                # 'range':[1000,1550.0],
                'tickmode':'auto',
                'showticklabels':True,
                'tickfont':{
                    'family':'Arial',
```

(continues on next page)

(continued from previous page)

```
                    'size':22,
                    'color':'rgb(0, 0, 0)',
                },
                'showexponent':'all',
                'exponentformat':'B',
                'ticklen':5,
                'tickwidth':1,
                'tickcolor':'rgb(0, 0, 0)',
                'ticks':'outside',
                'showline':True,
                'linecolor':'rgb(0, 0, 0)',
                'showgrid':False,
                'dtick':100,
            },
            'yaxis':{
                'type':'linear',
                'color':'rgb(0, 0, 0)',
                'title':'Intensity',
                'title_font':{
                    'family':'Arial',
                    'size':26,
                    'color':'rgb(0, 0, 0)',
                },
                'autorange':True,
                # 'range':[0.0,25.0],
                'showticklabels':True,
                'tickfont':{
                    'family':'Arial',
                    'size':22,
                    'color':'rgb(0, 0, 0)',
                },
                'tickangle':0,
                'showexponent':'all',
                'exponentformat':'B',
                'ticklen':5,
                'tickwidth':1,
                'tickcolor':'rgb(0, 0, 0)',
                'ticks':'outside',
                'showline':True,
                'linecolor':'rgb(0, 0, 0)',
                'showgrid':False,
                'zeroline':False,
            },
            # 'legend':{
            #     'font':{
            #         'family':'Arial',
            #         'size':10,
            #         'color':'rgb(0, 0, 0)',
            #     },
            #     'orientation':'v',
            #     'traceorder':'normal',
            # },
            'showlegend':False,
            'paper_bgcolor':'rgba(0,0,0,0)',
            'plot_bgcolor':'rgba(0,0,0,0)',
        },
    })
```

(continues on next page)

(continued from previous page)

```
    sugarpy_plots = uc.execute_misc_engine(
        input_file=spec_file,
        engine='sugarpy_plot_1_0_0',
        force = True,
    )

if __name__ == '__main__':
    main(
        spec_file=sys.argv[1],
        sugarpy_results=sys.argv[2],
    )
```

7.1.8 Cascade Search

Cascade search example

cascade_search_example.**search**(*validation_engine*)

Executes a cascade search on four example files from the data from Barth et al.

usage: ./cascade_search_example.py

Searches for peptides using a cascade search approach similar to Kertesz-Farkas et al. for which spectra were first searched for unmodified peptides, followed by consecutive searches for the following modifications: oxidation of M, deamidation of N/Q, methylation of E/K/R, N-terminal acetylation, phosphorylation of S/T. After each step, spectra with a PSM below 1 % PEP were removed.

cascade_search_example.**analyze**(*collector*)

Simle analysis script for the cascade search, counting the number of identified peptides (combination of peptide sequence and modifications) and PSMs (additionally include the spectrum ID)

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import csv
from collections import defaultdict as ddict
import os
import glob
import math

params = {
    "database": os.path.join(
        os.pardir,
        "example_data",
        "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
    ),
    "csv_filter_rules": [
        ["Is decoy", "equals", "false"],
        ["PEP", "lte", 0.01],
    ],
}
# We specify all search engines and validation engines that we want
# to use in a list (version numbers might differ on windows or mac):
search_engines = [
    "omssa",
```

(continues on next page)

(continued from previous page)

```
    "xtandem_piledriver",
    "msgfplus_v9979",
    # 'myrimatch_2_1_138',
    # 'msamanda_1_0_0_5243',
]

validation_engines = [
    "percolator_2_08",
    "qvality",
]

# The different levels with different modifications
# for the cascade are defined
cascade = {
    "0": ["C,fix,any,Carbamidomethyl"],
    "1": ["C,fix,any,Carbamidomethyl", "N,opt,any,Deamidated", "Q,opt,any,Deamidated
↪"],
    "2": ["C,fix,any,Carbamidomethyl", "*,opt,Prot-N-term,Acetyl"],
    "3": ["C,fix,any,Carbamidomethyl", "M,opt,any,Oxidation"],
    "4": [
        "C,fix,any,Carbamidomethyl",
        "E,opt,any,Methyl",
        "K,opt,any,Methyl",
        "R,opt,any,Methyl",
    ],
    "5": ["C,fix,any,Carbamidomethyl", "S,opt,any,Phospho", "T,opt,any,Phospho"],
}

mass_spectrometer = "LTQ XL low res"

get_params = {
    "ftp_url": "ftp.peptideatlas.org",
    "ftp_login": "PASS00269",
    "ftp_password": "FI4645a",
    "ftp_include_ext": [
        "JB_FASP_pH8_2-3_28122012.mzML",
        "JB_FASP_pH8_2-4_28122012.mzML",
        "JB_FASP_pH8_3-1_28122012.mzML",
        "JB_FASP_pH8_4-1_28122012.mzML",
    ],
    "ftp_output_folder": os.path.join(os.pardir, "example_data", "cascade_search"),
    "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_with_
↪contaminants_target_decoy.fasta",
    "http_output_folder": os.path.join(os.pardir, "example_data"),
}


def get_files():
    uc = ursgal.UController(params=get_params)
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")

    if os.path.exists(get_params["ftp_output_folder"]) is False:
        os.makedirs(get_params["ftp_output_folder"])
    uc.fetch_file(engine="get_ftp_files_1_0_0")

    spec_files = []
```

(continues on next page)

(continued from previous page)

```
    for mzML_file in glob.glob(os.path.join(get_params["ftp_output_folder"], "*.mzML
↪")):
        spec_files.append(mzML_file)

    return spec_files


def search(validation_engine):
    """
    Executes a cascade search on four example files from the
    data from Barth et al.

    usage:
        ./cascade_search_example.py

    Searches for peptides using a cascade search approach similar to Kertesz-Farkas␣
↪et al.
    for which spectra were first searched for unmodified peptides, followed by␣
↪consecutive searches
    for the following modifications:
    oxidation of M,
    deamidation of N/Q,
    methylation of E/K/R,
    N-terminal acetylation,
    phosphorylation of S/T.
    After each step, spectra with a PSM below 1 % PEP were removed.
    """
    # Initializing the uPLANIT UController class with
    # our specified modifications and mass spectrometer
    uc = ursgal.UController(
        profile=mass_spectrometer, params=params  # 'LTQ XL low res' profile!
    )
    # complete workflow for every level of the cascade:
    # every spectrum file is searched with every search engine,
    # results are validated seperately,
    # validated results are merged and filtered for targets and PEP <= 0.01.

    def workflow(
        spec_file,
        prefix=None,
        validation_engine=None,
        filter_before_validation=False,
        force=False,
    ):
        validated_results = []

        # Convert mzML to MGF outside the loop, so this step is not repeated in
        # the loop
        mgf_spec_file = uc.convert(input_file=spec_file, engine="mzml2mgf_1_0_0")
        for search_engine in search_engines:
            uc.params["prefix"] = prefix
            unified_search_results = uc.search(
                input_file=mgf_spec_file,
                engine=search_engine,
                force=force,
            )
            uc.params["prefix"] = ""
```

(continues on next page)

(continued from previous page)

```
            if filter_before_validation == True:
                uc.params["csv_filter_rules"] = [
                    [
                        "Modifications",
                        "contains",
                        "{0}".format(cascade[level][1].split(",")[3]),
                    ]
                ]
                filtered_search_results = uc.execute_misc_engine(
                    input_file=unified_search_results, engine="filter_csv_1_0_0"
                )
            else:
                filtered_search_results = unified_search_results
            validated_search_results = uc.validate(
                input_file=filtered_search_results,
                engine=validation_engine,
                force=force,
            )
            validated_results.append(validated_search_results)

        validated_results_from_all_engines = uc.execute_misc_engine(
            input_file=sorted(validated_results),
            engine="merge_csvs_1_0_0",
            force=force,
        )
        uc.params["csv_filter_rules"] = [
            ["Is decoy", "equals", "false"],
            ["PEP", "lte", 0.01],
        ]
        filtered_validated_results = uc.execute_misc_engine(
            input_file=validated_results_from_all_engines, engine="filter_csv_1_0_0"
        )
        return filtered_validated_results

    result_files = []
    for spec_file in spec_files:
        spectra_with_PSM = set()
        for level in sorted(cascade.keys()):
            uc.params["modifications"] = cascade[level]
            if level == "0":
                results = workflow(
                    spec_file,
                    validation_engine=validation_engine,
                    prefix="cascade-lvl-{0}".format(level),
                )
            else:
                uc.params["scan_exclusion_list"] = list(spectra_with_PSM)
                results = workflow(
                    spec_file,
                    validation_engine=validation_engine,
                    filter_before_validation=True,
                    force=True,
                    prefix="cascade-lvl-{0}".format(level),
                )
            result_files.append(results)
            #  spectrum IDs for PSMs are written into an exclusion list for the next␣
↪level of the cascade search,
```

(continues on next page)

(continued from previous page)

```
            #  these spectra will b excluded during mzml2mgf conversion
            with open(results) as in_file:
                csv_input = csv.DictReader(in_file)
                for line_dict in csv_input:
                    spectra_with_PSM.add(line_dict["Spectrum ID"])
            print(
                "Number of spectra that will be removed for the next cacade level: {0}
↪".format(
                    len(spectra_with_PSM)
                )
            )

    results_all_files = uc.execute_misc_engine(
        input_file=sorted(result_files),
        engine="merge_csvs_1_0_0",
    )
    return results_all_files


def analyze(collector):
    """
    Simle analysis script for the cascade search,
    counting the number of identified peptides (combination of peptide sequence and␣
↪modifications)
    and PSMs (additionally include the spectrum ID)
    """

    mod_list = ["Oxidation", "Deamidated", "Methyl", "Acetyl", "Phospho"]
    fieldnames = (
        ["approach", "count_type", "validation_engine", "unmodified", "multimodified"]
        + mod_list
        + ["total"]
    )

    csv_writer = csv.DictWriter(open("cascade_results.csv", "w"), fieldnames)
    csv_writer.writeheader()
    uc = ursgal.UController()
    uc.params["validation_score_field"] = "PEP"
    uc.params["bigger_scores_better"] = False

    # Count the number of identified peptides and PSMs for the different modifications
    # Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score␣
↪is counted
    # and only if the best hit has a PEP that is at least two orders of
    # magnitude smaller than the others
    for validation_engine, result_file in collector.items():
        counter_dict = {"psm": ddict(set), "pep": ddict(set)}
        grouped_psms = uc._group_psms(
            result_file, validation_score_field="PEP", bigger_scores_better=False
        )
        for spec_title, grouped_psm_list in grouped_psms.items():
            best_score, best_line_dict = grouped_psm_list[0]
            if len(grouped_psm_list) > 1:
                second_best_score, second_best_line_dict = grouped_psm_list[1]
                best_peptide_and_mod = (
                    best_line_dict["Sequence"] + best_line_dict["Modifications"]
                )
```

(continues on next page)

(continued from previous page)

```
                second_best_peptide_and_mod = (
                    second_best_line_dict["Sequence"]
                    + second_best_line_dict["Modifications"]
                )

                if best_peptide_and_mod == second_best_peptide_and_mod:
                    line_dict = best_line_dict
                elif best_line_dict["Sequence"] == second_best_line_dict["Sequence"]:
                    if best_score == second_best_score:
                        line_dict = best_line_dict
                    else:
                        if (-1 * math.log10(best_score)) - (
                            -1 * math.log10(second_best_score)
                        ) >= 2:
                            line_dict = best_line_dict
                        else:
                            continue
                else:
                    if (-1 * math.log10(best_score)) - (
                        -1 * math.log10(second_best_score)
                    ) >= 2:
                        line_dict = best_line_dict
                    else:
                        continue
            else:
                line_dict = best_line_dict

            count = 0
            for mod in mod_list:
                if mod in line_dict["Modifications"]:
                    count += 1
            key_2_add = ""
            if count == 0:
                key_2_add = "unmodified"
            elif count >= 2:
                key_2_add = "multimodified"
            elif count == 1:
                for mod in mod_list:
                    if mod in line_dict["Modifications"]:
                        key_2_add = mod
                        break
            # for peptide identification comparison
            counter_dict["pep"][key_2_add].add(
                line_dict["Sequence"] + line_dict["Modifications"]
            )
            # for PSM comparison
            counter_dict["psm"][key_2_add].add(
                line_dict["Spectrum Title"]
                + line_dict["Sequence"]
                + line_dict["Modifications"]
            )
        for counter_key, count_dict in counter_dict.items():
            dict_2_write = {
                "approach": "cascade",
                "count_type": counter_key,
                "validation_engine": validation_engine,
            }
```

(continues on next page)

(continued from previous page)

```
            total_number = 0
            for key, obj_set in count_dict.items():
                dict_2_write[key] = len(obj_set)
                total_number += len(obj_set)
            dict_2_write["total"] = total_number
            csv_writer.writerow(dict_2_write)
    return


if __name__ == "__main__":
    spec_files = get_files()
    collector = {}
    for validation_engine in validation_engines:
        results_all_files = search(validation_engine)
        print(
            ">>> ",
            "final results for {0}".format(validation_engine),
            " were written into:",
        )
        print(">>> ", results_all_files)
        collector[validation_engine] = results_all_files
    analyze(collector)
    print(">>> ", "number of identified peptides and PSMs were written into:")
    print(">>> ", "cascade_results.csv")
```

Ungrouped search for comparison

ungrouped_search_example.**search**(*validation_engine*)

Executes an ungrouped search on four example files from the data from Barth et al.

usage: ./ungrouped_search_example.py

Searches for peptides including the following potential modifications: oxidation of M, deamidation of N/Q, methylation of E/K/R, N-terminal acetylation, phosphorylation of S/T.

All modifications are validated together with unmodified peptides.

ungrouped_search_example.**analyze**(*collector*)

Simple analysis script for the ungrouped search, counting the number of identified peptides (combination of peptide sequence and modifications) and PSMs (additionally include the spectrum ID)

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import csv
from collections import defaultdict as ddict
import os
import glob
import math
import sys

params = {
    "database": os.path.join(
        os.pardir,
        "example_data",
        "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
```

(continues on next page)

(continued from previous page)

```
    ),
    "csv_filter_rules": [
        ["Is decoy", "equals", "false"],
        ["PEP", "lte", 0.01],
    ],
    # Modifications that should be included in the search
    "modifications": [
        "C,fix,any,Carbamidomethyl",
        "M,opt,any,Oxidation",
        "N,opt,any,Deamidated",
        "Q,opt,any,Deamidated",
        "E,opt,any,Methyl",
        "K,opt,any,Methyl",
        "R,opt,any,Methyl",
        "*,opt,Prot-N-term,Acetyl",
        "S,opt,any,Phospho",
        "T,opt,any,Phospho",
    ],
}
# We specify all search engines and validation engines that we want
# to use in a list (version numbers might differ on windows or mac):
search_engines = [
    "omssa",
    "xtandem_piledriver",
    "msgfplus_v9979",
    # 'myrimatch_2_1_138',
    "msamanda_1_0_0_5243",
]
validation_engines = [
    "percolator_2_08",
    "qvality",
]

mass_spectrometer = "LTQ XL low res"

get_params = {
    "ftp_url": "ftp.peptideatlas.org",
    "ftp_login": "PASS00269",
    "ftp_password": "FI4645a",
    "ftp_include_ext": [
        "JB_FASP_pH8_2-3_28122012.mzML",
        "JB_FASP_pH8_2-4_28122012.mzML",
        "JB_FASP_pH8_3-1_28122012.mzML",
        "JB_FASP_pH8_4-1_28122012.mzML",
    ],
    "ftp_output_folder": os.path.join(os.pardir, "example_data", "ungrouped_search"),
    "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_with_
↪contaminants_target_decoy.fasta",
    "http_output_folder": os.path.join(os.pardir, "example_data"),
}


def get_files():
    uc = ursgal.UController(params=get_params)

    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")
```

(continues on next page)

(continued from previous page)

```
    if os.path.exists(get_params["ftp_output_folder"]) is False:
        os.makedirs(get_params["ftp_output_folder"])
    uc.fetch_file(engine="get_ftp_files_1_0_0")

    spec_files = []
    for mzML_file in glob.glob(os.path.join(get_params["ftp_output_folder"], "*.mzML
↪")):
        spec_files.append(mzML_file)

    return spec_files


def search(validation_engine):
    """
    Executes an ungrouped search on four example files from the
    data from Barth et al.

    usage:
        ./ungrouped_search_example.py

    Searches for peptides including the following potential modifications:
    oxidation of M,
    deamidation of N/Q,
    methylation of E/K/R,
    N-terminal acetylation,
    phosphorylation of S/T.

    All modifications are validated together with unmodified peptides.
    """

    uc = ursgal.UController(
        profile=mass_spectrometer, params=params  # 'LTQ XL low res' profile!
    )

    # complete workflow:
    # every spectrum file is searched with every search engine,
    # results are validated seperately,
    # validated results are merged and filtered for targets and PEP <= 0.01.
    # In the end, all filtered results from all spectrum files are merged
    # for validation_engine in validation_engines:
    result_files = []
    for spec_file in spec_files:
        validated_results = []
        for search_engine in search_engines:
            unified_search_results = uc.search(
                input_file=spec_file,
                engine=search_engine,
            )
            validated_search_results = uc.validate(
                input_file=unified_search_results,
                engine=validation_engine,
            )
            validated_results.append(validated_search_results)

        validated_results_from_all_engines = uc.execute_misc_engine(
            input_file=sorted(validated_results),
            engine="merge_csvs",
```

(continues on next page)

(continued from previous page)

```
        )
        filtered_validated_results = uc.execute_misc_engine(
            input_file=validated_results_from_all_engines, engine="filter_csv"
        )
        result_files.append(filtered_validated_results)

    results_all_files = uc.execute_misc_engine(
        input_file=sorted(result_files),
        engine="merge_csvs",
    )
    return results_all_files


def analyze(collector):
    """
    Simple analysis script for the ungrouped search,
    counting the number of identified peptides (combination of peptide sequence and␣
↪modifications)
    and PSMs (additionally include the spectrum ID)
    """
    mod_list = ["Oxidation", "Deamidated", "Methyl", "Acetyl", "Phospho"]
    fieldnames = (
        ["approach", "count_type", "validation_engine", "unmodified", "multimodified"]
        + mod_list
        + ["total"]
    )

    csv_writer = csv.DictWriter(open("ungrouped_results.csv", "w"), fieldnames)
    csv_writer.writeheader()
    uc = ursgal.UController()
    uc.params["validation_score_field"] = "PEP"
    uc.params["bigger_scores_better"] = False

    # Count the number of identified peptides and PSMs for the different modifications
    # Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score␣
↪is counted
    # and only if the best hit has a PEP that is at least two orders of
    # magnitude smaller than the others
    for validation_engine, result_file in collector.items():
        counter_dict = {"psm": ddict(set), "pep": ddict(set)}
        grouped_psms = uc._group_psms(
            result_file, validation_score_field="PEP", bigger_scores_better=False
        )
        for spec_title, grouped_psm_list in grouped_psms.items():
            best_score, best_line_dict = grouped_psm_list[0]
            if len(grouped_psm_list) > 1:
                second_best_score, second_best_line_dict = grouped_psm_list[1]
                best_peptide_and_mod = (
                    best_line_dict["Sequence"] + best_line_dict["Modifications"]
                )
                second_best_peptide_and_mod = (
                    second_best_line_dict["Sequence"]
                    + second_best_line_dict["Modifications"]
                )

                if best_peptide_and_mod == second_best_peptide_and_mod:
                    line_dict = best_line_dict
```

(continues on next page)

(continued from previous page)

```
                elif best_line_dict["Sequence"] == second_best_line_dict["Sequence"]:
                    if best_score == second_best_score:
                        line_dict = best_line_dict
                    else:
                        if (-1 * math.log10(best_score)) - (
                            -1 * math.log10(second_best_score)
                        ) >= 2:
                            line_dict = best_line_dict
                        else:
                            continue
                else:
                    if (-1 * math.log10(best_score)) - (
                        -1 * math.log10(second_best_score)
                    ) >= 2:
                        line_dict = best_line_dict
                    else:
                        continue
            else:
                line_dict = best_line_dict

            count = 0
            for mod in mod_list:
                if mod in line_dict["Modifications"]:
                    count += 1
            key_2_add = ""
            if count == 0:
                key_2_add = "unmodified"
            elif count >= 2:
                key_2_add = "multimodified"
            elif count == 1:
                for mod in mod_list:
                    if mod in line_dict["Modifications"]:
                        key_2_add = mod
                        break
            # for peptide identification comparison
            counter_dict["pep"][key_2_add].add(
                line_dict["Sequence"] + line_dict["Modifications"]
            )
            # for PSM comparison
            counter_dict["psm"][key_2_add].add(
                line_dict["Spectrum Title"]
                + line_dict["Sequence"]
                + line_dict["Modifications"]
            )
        for counter_key, count_dict in counter_dict.items():
            dict_2_write = {
                "approach": "ungrouped",
                "count_type": counter_key,
                "validation_engine": validation_engine,
            }
            total_number = 0
            for key, obj_set in count_dict.items():
                dict_2_write[key] = len(obj_set)
                total_number += len(obj_set)
            dict_2_write["total"] = total_number
            csv_writer.writerow(dict_2_write)
    return
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    spec_files = get_files()
    collector = {}
    for validation_engine in validation_engines:
        results_all_files = search(validation_engine)
        print(
            ">>> ",
            "final results for {0}".format(validation_engine),
            " were written into:",
        )
        print(">>> ", results_all_files)
        collector[validation_engine] = results_all_files
    analyze(collector)
    print(">>> ", "number of identified peptides and PSMs were written into:")
    print(">>> ", "ungrouped_results.csv")
```

Grouped search for comparison

grouped_search_example.**search**(*validation_engine*)

Executes a grouped search on four example files from the data from Barth et al.

usage: ./grouped_search_example.py

Searches for peptides including the following potential modifications: oxidation of M, deamidation of N/Q, methylation of E/K/R, N-terminal acetylation, phosphorylation of S/T.

After the search, each type of modification is validated seperately.

grouped_search_example.**analyze**(*collector*)

Simle analysis script for the grouped search, counting the number of identified peptides (combination of peptide sequence and modifications) and PSMs (additionally include the spectrum ID)

```
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import csv
from collections import defaultdict as ddict
import os
import glob
import math
from collections import defaultdict as ddict

params = {
    "database": os.path.join(
        os.pardir,
        "example_data",
        "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
    ),
    "csv_filter_rules": [
        ["Is decoy", "equals", "false"],
        ["PEP", "lte", 0.01],
    ],
    # Modifications that should be included in the search
    "modifications": [
```

(continues on next page)

(continued from previous page)

```
        "C,fix,any,Carbamidomethyl",
        "M,opt,any,Oxidation",
        "N,opt,any,Deamidated",
        "Q,opt,any,Deamidated",
        "E,opt,any,Methyl",
        "K,opt,any,Methyl",
        "R,opt,any,Methyl",
        "*,opt,Prot-N-term,Acetyl",
        "S,opt,any,Phospho",
        "T,opt,any,Phospho",
    ],
}
# We specify all search engines and validation engines that we want
# to use in a list (version numbers might differ on windows or mac):
search_engines = [
    "omssa",
    "xtandem_piledriver",
    "msgfplus_v9979",
    # 'myrimatch_2_1_138',
    "msamanda_1_0_0_5243",
]
validation_engines = [
    "percolator_2_08",
    "qvality",
]

# Groups that are evaluated seperately
groups = {
    "0": "",
    "1": "Oxidation",
    "2": "Deamidated",
    "3": "Methyl",
    "4": "Acetyl",
    "5": "Phospho",
}

mass_spectrometer = "LTQ XL low res"

get_params = {
    "ftp_url": "ftp.peptideatlas.org",
    "ftp_login": "PASS00269",
    "ftp_password": "FI4645a",
    "ftp_include_ext": [
        "JB_FASP_pH8_2-3_28122012.mzML",
        "JB_FASP_pH8_2-4_28122012.mzML",
        "JB_FASP_pH8_3-1_28122012.mzML",
        "JB_FASP_pH8_4-1_28122012.mzML",
    ],
    "ftp_output_folder": os.path.join(os.pardir, "example_data", "grouped_search"),
    "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_with_
↪contaminants_target_decoy.fasta",
    "http_output_folder": os.path.join(os.pardir, "example_data"),
}


def get_files():
    uc = ursgal.UController(params=get_params)
```

(continues on next page)

(continued from previous page)

```
    if os.path.exists(params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")
    if os.path.exists(get_params["ftp_output_folder"]) is False:
        os.makedirs(get_params["ftp_output_folder"])
    uc.fetch_file(engine="get_ftp_files_1_0_0")

    spec_files = []
    for mzML_file in glob.glob(os.path.join(get_params["ftp_output_folder"], "*.mzML
↪")):
        spec_files.append(mzML_file)

    return spec_files


def search(validation_engine):
    """
    Executes a grouped search on four example files from the
    data from Barth et al.

    usage:
        ./grouped_search_example.py

    Searches for peptides including the following potential modifications:
    oxidation of M,
    deamidation of N/Q,
    methylation of E/K/R,
    N-terminal acetylation,
    phosphorylation of S/T.

    After the search, each type of modification is validated seperately.
    """
    # Initializing the ursgal UController class with
    # our specified modifications and mass spectrometer
    uc = ursgal.UController(
        profile=mass_spectrometer, params=params  # 'LTQ XL low res' profile!
    )

    # complete workflow:
    # every spectrum file is searched with every search engine,
    # results are seperated into groups and validated seperately,
    # validated results are merged and filtered for targets and PEP <= 0.01.
    # In the end, all filtered results from all spectrum files are merged
    # for validation_engine in validation_engines:
    result_files = []
    for n, spec_file in enumerate(spec_files):
        validated_results = []
        for search_engine in search_engines:
            unified_search_results = uc.search(
                input_file=spec_file,
                engine=search_engine,
            )

            # Calculate PEP for every group seperately, therefore need to split
            # the csv first
            group_list = sorted(groups.keys())
            for p, group in enumerate(group_list):
```

(continues on next page)

(continued from previous page)

```
                if group == "0":
                    uc.params["csv_filter_rules"] = [
                        ["Modifications", "contains_not", "{0}".format(groups["1"])],
                        ["Modifications", "contains_not", "{0}".format(groups["2"])],
                        ["Modifications", "contains_not", "{0}".format(groups["3"])],
                        ["Modifications", "contains_not", "{0}".format(groups["4"])],
                        ["Modifications", "contains_not", "{0}".format(groups["5"])],
                    ]
                else:
                    uc.params["csv_filter_rules"] = [
                        ["Modifications", "contains", "{0}".format(groups[group])]
                    ]
                    for other_group in group_list:
                        if other_group == "0" or other_group == group:
                            continue
                        uc.params["csv_filter_rules"].append(
                            [
                                "Modifications",
                                "contains_not",
                                "{0}".format(groups[other_group]),
                            ],
                        )
                uc.params["prefix"] = "grouped-{0}".format(group)
                filtered_results = uc.execute_misc_engine(
                    input_file=unified_search_results, engine="filter_csv"
                )
                uc.params["prefix"] = ""
                validated_search_results = uc.validate(
                    input_file=filtered_results,
                    engine=validation_engine,
                )
                validated_results.append(validated_search_results)

        uc.params["prefix"] = "file{0}".format(n)
        validated_results_from_all_engines = uc.execute_misc_engine(
            input_file=sorted(validated_results),
            engine="merge_csvs",
        )
        uc.params["prefix"] = ""
        uc.params["csv_filter_rules"] = [
            ["Is decoy", "equals", "false"],
            ["PEP", "lte", 0.01],
        ]
        filtered_validated_results = uc.execute_misc_engine(
            input_file=validated_results_from_all_engines, engine="filter_csv"
        )
        result_files.append(filtered_validated_results)

    results_all_files = uc.execute_misc_engine(
        input_files=sorted(result_files),
        engine="merge_csvs",
    )
    return results_all_files


def analyze(collector):
    """
```

(continues on next page)

(continued from previous page)

```
    Simle analysis script for the grouped search,
    counting the number of identified peptides (combination of peptide sequence and␣
↪modifications)
    and PSMs (additionally include the spectrum ID)
    """
    mod_list = ["Oxidation", "Deamidated", "Methyl", "Acetyl", "Phospho"]
    fieldnames = (
        ["approach", "count_type", "validation_engine", "unmodified", "multimodified"]
        + mod_list
        + ["total"]
    )

    csv_writer = csv.DictWriter(open("grouped_results.csv", "w"), fieldnames)
    csv_writer.writeheader()
    uc = ursgal.UController()
    uc.params["validation_score_field"] = "PEP"
    uc.params["bigger_scores_better"] = False

    # Count the number of identified peptides and PSMs for the different modifications
    # Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score␣
↪is counted
    # and only if the best hit has a PEP that is at least two orders of
    # magnitude smaller than the others
    for validation_engine, result_file in collector.items():
        counter_dict = {"psm": ddict(set), "pep": ddict(set)}
        grouped_psms = uc._group_psms(
            result_file, validation_score_field="PEP", bigger_scores_better=False
        )
        for spec_title, grouped_psm_list in grouped_psms.items():
            best_score, best_line_dict = grouped_psm_list[0]
            if len(grouped_psm_list) > 1:
                second_best_score, second_best_line_dict = grouped_psm_list[1]
                best_peptide_and_mod = (
                    best_line_dict["Sequence"] + best_line_dict["Modifications"]
                )
                second_best_peptide_and_mod = (
                    second_best_line_dict["Sequence"]
                    + second_best_line_dict["Modifications"]
                )

                if best_peptide_and_mod == second_best_peptide_and_mod:
                    line_dict = best_line_dict
                elif best_line_dict["Sequence"] == second_best_line_dict["Sequence"]:
                    if best_score == second_best_score:
                        line_dict = best_line_dict
                    else:
                        if (-1 * math.log10(best_score)) - (
                            -1 * math.log10(second_best_score)
                        ) >= 2:
                            line_dict = best_line_dict
                        else:
                            continue
                else:
                    if (-1 * math.log10(best_score)) - (
                        -1 * math.log10(second_best_score)
                    ) >= 2:
                        line_dict = best_line_dict
```

(continues on next page)

(continued from previous page)

```
                    else:
                        continue
            else:
                line_dict = best_line_dict

            count = 0
            for mod in mod_list:
                if mod in line_dict["Modifications"]:
                    count += 1
            key_2_add = ""
            if count == 0:
                key_2_add = "unmodified"
            elif count >= 2:
                key_2_add = "multimodified"
            elif count == 1:
                for mod in mod_list:
                    if mod in line_dict["Modifications"]:
                        key_2_add = mod
                        break
            # for peptide identification comparison
            counter_dict["pep"][key_2_add].add(
                line_dict["Sequence"] + line_dict["Modifications"]
            )
            # for PSM comparison
            counter_dict["psm"][key_2_add].add(
                line_dict["Spectrum Title"]
                + line_dict["Sequence"]
                + line_dict["Modifications"]
            )
        for counter_key, count_dict in counter_dict.items():
            dict_2_write = {
                "approach": "grouped",
                "count_type": counter_key,
                "validation_engine": validation_engine,
            }
            total_number = 0
            for key, obj_set in count_dict.items():
                dict_2_write[key] = len(obj_set)
                total_number += len(obj_set)
            dict_2_write["total"] = total_number
            csv_writer.writerow(dict_2_write)
    return


if __name__ == "__main__":
    spec_files = get_files()
    collector = {}
    for validation_engine in validation_engines:
        results_all_files = search(validation_engine)
        print(
            ">>> ",
            "final results for {0}".format(validation_engine),
            " were written into:",
        )
        print(">>> ", results_all_files)
        collector[validation_engine] = results_all_files
    analyze(collector)
```

(continues on next page)

(continued from previous page)

```
    print(">>> ", "number of identified peptides and PSMs were written into:")
    print(">>> ", "grouped_results.csv")
```

Example results for cascade search

ROS dataset results from a subsset of data from Barth et al. 2014.

approach	count_type	validation_engine	unmodified	multi-modified	Oxidation	Deamidated	Methyl	Acetyl	Phospho	total
ungrouped	psm	qvality	6905	24	270	157	42	23	6	7427
grouped	psm	qvality	8009		267	128	31	27	3	8465
cascade	psm	qvality	8009		316	131	46	30	1	7788
ungrouped	upep	qvality	1409	14	94	48	20	11	3	1599
grouped	upep	qvality	1575		93	42	9	17	1	1737
cascade	upep	qvality	1552		102	44	13	13	1	1725

Human BR dataset results.

approach	count_type	validation_engine	unmodified	multi-modified	Oxidation	Deamidated	Methyl	Acetyl	Phospho	total
ungrouped	psm	qvality	18813	51	332	336	102	292	122	20048
grouped	psm	qvality	21370		383	171	100	435	142	22601
cascade	psm	qvality	19065		424	251	111	417	38	20406
ungrouped	upep	qvality	6707	39	143	173	27	123	58	7270
grouped	upep	qvality	7525		161	104	31	191	58	8070
cascade	upep	qvality	7784		172	152	43	171	55	8377

7.1.9 Upeptide Mapper Example Scripts

Upeptide mapper v3 and 4 complete C. reinhardtii proteome match

complete_chlamydomonas_proteome_match.**main**(*class_version*)

Example script to demonstrate speed and memory efficiency of the new upeptide_mapper.

All tryptic peptides (n=1,094,395, 6 < len(peptide) < 40) are mapped to the Chlamydomonas reinhardtii (38876 entries) target-decoy database.

usage: ./complete_chlamydomonas_proteome_match.py <class_version>

Class versions

- UPeptideMapper_v2
- UPeptideMapper_v3
- UPeptideMapper_v4

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


import ursgal
import glob
import os.path
import sys
import time


def main(class_version):
    """

    Example script to demonstrate speed and memory efficiency of the new
    upeptide_mapper.

    All tryptic peptides (n=1,094,395, 6 < len(peptide) < 40 ) are mapped to the
    Chlamydomonas reinhardtii (38876 entries) target-decoy database.

    usage:
        ./complete_chlamydomonas_proteome_match.py <class_version>

    Class versions
        * UPeptideMapper_v2
        * UPeptideMapper_v3
        * UPeptideMapper_v4

    """

    input_params = {
        "database": os.path.join(
            os.pardir,
            "example_data",
            "Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta",
        ),
        "http_url": "https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
↪with_contaminants_target_decoy.fasta",
        "http_output_folder": os.path.join(
            os.pardir,
            "example_data",
        ),
    }

    uc = ursgal.UController(params=input_params)

    if os.path.exists(input_params["database"]) is False:
        uc.fetch_file(engine="get_http_files_1_0_0")
    print("Parsing fasta and digesting sequences")
    peptides = set()
    digest_start = time.time()
    for fastaID, sequence in ursgal.ucore.parse_fasta(
        open(input_params["database"], "r")
```

(continues on next page)

(continued from previous page)

```
    ):
        tryptic_peptides = ursgal.ucore.digest(
            sequence, ("KR", "C"), no_missed_cleavages=True
        )
        for p in tryptic_peptides:
            if 6 <= len(p) <= 40:
                peptides.add(p)
    print(
        "Parsing fasta and digesting sequences took {0:1.2f} seconds".format(
            time.time() - digest_start
        )
    )
    if sys.platform == "win32":
        print(
            "[ WARNING ] pyahocorasick can not be installed via pip on Windwows at␣
↪the moment\n"
            "[ WARNING ] Falling back to UpeptideMapper_v2"
        )
        class_version = "UPeptideMapper_v2"

    upapa_class = uc.unodes["upeptide_mapper_1_0_0"][
        "class"
    ].import_engine_as_python_function(class_version)

    print("Buffering fasta and mapping {0} peptides".format(len(peptides)))
    map_start = time.time()

    if class_version == "UPeptideMapper_v2":
        peptide_mapper = upapa_class(word_len=6)
        fasta_lookup_name = peptide_mapper.build_lookup_from_file(
            input_params["database"],
            force=False,
        )
        args = [list(peptides), fasta_lookup_name]
    elif class_version == "UPeptideMapper_v3":
        peptide_mapper = upapa_class(input_params["database"])
        fasta_lookup_name = peptide_mapper.fasta_name
        args = [list(peptides), fasta_lookup_name]
    elif class_version == "UPeptideMapper_v4":
        peptide_mapper = upapa_class(input_params["database"])
        args = [list(peptides)]

    p2p_mappings = peptide_mapper.map_peptides(*args)
    print(
        "Buffering fasta and mapping {0} peptides took {1:1.2f} seconds".format(
            len(peptides), time.time() - map_start
        )
    )
    if len(p2p_mappings.keys()) == len(peptides):
        print("All peptides have been mapped!")
    else:
        print("WARNING: Not all peptide have been mapped")


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print(main.__doc__)
```

(continues on next page)

(continued from previous page)

```
        sys.exit(1)
    main(sys.argv[1])
```

Upeptide mapper v3 and 4 complete proteome match

complete_proteome_match.**main**(*fasta_database*, *class_version*)

Example script to demonstrate speed and memory efficiency of the new upeptide_mapper.

Specify fasta_database and class_version as input.

usage: ./complete_proteome_match.py <fasta_database> <class_version>

Class versions

- UPeptideMapper_v2
- UPeptideMapper_v3
- UPeptideMapper_v4

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


import ursgal
import glob
import os.path
import sys
import time


def main(fasta_database, class_version):
    """

    Example script to demonstrate speed and memory efficiency of the new
    upeptide_mapper.

    Specify fasta_database and class_version as input.

    usage:
        ./complete_proteome_match.py <fasta_database> <class_version>

    Class versions
        * UPeptideMapper_v2
        * UPeptideMapper_v3
        * UPeptideMapper_v4

    """

    input_params = {
        "database": sys.argv[1],
    }

    uc = ursgal.UController(params=input_params)

    print("Parsing fasta and digesting sequences")
    peptides = set()
```

(continues on next page)

(continued from previous page)

```
    digest_start = time.time()
    for fastaID, sequence in ursgal.ucore.parse_fasta(
        open(input_params["database"], "r")
    ):
        tryptic_peptides = ursgal.ucore.digest(
            sequence,
            ("KR", "C"),
            # no_missed_cleavages = True
        )
        for p in tryptic_peptides:
            if 6 <= len(p) <= 40:
                peptides.add(p)
    print(
        "Parsing fasta and digesting sequences took {0:1.2f} seconds".format(
            time.time() - digest_start
        )
    )

    if sys.platform == "win32":
        print(
            "[ WARNING ] pyahocorasick can not be installed via pip on Windwows at␣
↪the moment\n"
            "[ WARNING ] Falling back to UpeptideMapper_v2"
        )
        class_version = "UPeptideMapper_v2"

    upapa_class = uc.unodes["upeptide_mapper_1_0_0"][
        "class"
    ].import_engine_as_python_function(class_version)

    print("Buffering fasta and mapping {0} peptides".format(len(peptides)))
    map_start = time.time()

    if class_version == "UPeptideMapper_v2":
        peptide_mapper = upapa_class(word_len=6)
        fasta_lookup_name = peptide_mapper.build_lookup_from_file(
            input_params["database"],
            force=False,
        )
        args = [list(peptides), fasta_lookup_name]
    elif class_version == "UPeptideMapper_v3":
        peptide_mapper = upapa_class(input_params["database"])
        fasta_lookup_name = peptide_mapper.fasta_name
        args = [list(peptides), fasta_lookup_name]
    elif class_version == "UPeptideMapper_v4":
        peptide_mapper = upapa_class(input_params["database"])
        args = [list(peptides)]
    p2p_mappings = peptide_mapper.map_peptides(*args)
    print(
        "Buffering fasta and mapping {0} peptides took {1:1.2f} seconds".format(
            len(peptides), time.time() - map_start
        )
    )
    if len(p2p_mappings.keys()) == len(peptides):
        print("All peptides have been mapped!")
    else:
        print("WARNING: Not all peptide have been mapped")
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    if len(sys.argv) < 3:
        print(main.__doc__)
        sys.exit(1)
    main(sys.argv[1], sys.argv[2])
```

CHAPTER 8

Changelog

Record of released Ursgal versions with all notable changes

8.1 Changelog

8.1.1 Version 0.6.8 (12.2020)

1. Simplified the isntallation of third-party engines

8.1.2 Version 0.6.7 (11.2020)

1. Implemented pNovo 3.1.3
2. Implemented pGlyco 2.2.2
3. Implemented Percolator 3.4.0
4. Implemented PTM-Shepherd 0.3.5
5. Implemented TagGraph 1.8.0
6. Implemented latest version of MODa (v1.62), MSAmanda (version 2.0.0.14665), MSFragger (20190628, 2.3, 3.0), DeepNovo (PointNovo)

*. Implemented glycopeptide search functionality for MSFragger 3.0 #. Implemented MS-GF+ v2019.07.03 and included enzymes.txt for more and customized options #. PSM defining columns for sanitize_csv and combine_PEP are now

specified though uparams

1. Added advanced protein digest functionality (to ucore) adapted from Pyteomics' cleave function
2. venndiagram_1_1_0 can now also print percentages for each field
3. Added signal_to_noise_threshold option for mzml2mgf conversion

4. internal changes to the mapping of styles and uparams in umapmaster

8.1.3 Version 0.6.6 (03.2020)

1. Setup.py (and pip) installs resources now
2. Added Percolator 3.4
3. Dropped pymzml v0.7.x support - mzml2mgf v2.0.0 is default. If pymzml v.07 is required change default mzml2mgf converter back to 1.0.0
4. Dropped Python 3.4 support

8.1.4 Version 0.6.5 (05.2019)

1. Added ThermoRawFileParser for conversion of .raw to .mzML/.mgf
2. Added pGlyco 2.2.0, including pParse 2.0 and pGlycoFDR 2.2.0
3. Added DeepNovo 0.0.1 (so far only search_denovo functionality)
4. Added latest version of MSGF+ (version 2019.04.18) and MSFragger (version 20190222)

8.1.5 Version 0.6.4 (05.2019)

1. Fixed bug in unify csv where N-terminal mods were not assigned the correct position
2. Added possibility to write result groups of venndiagram to csv
3. Use setuptools instead of distutils in setup.py
4. Allowed installation via PyPI

8.1.6 Version 0.6.3 (03.2019)

1. Added Percolator 3.2, including the options to infer protein PEP and FDR
2. Added latest versions of MSGF+ (version 2019.01.22) and MSAmanda (version 2.0.0.11219) and Novor (version 1.05)
3. Added support for conversion of mzML files originating from .wiff files based on the id_dict property in the latest pymzML version (2.1.0)
4. Some minor bugfixes and improvements (check git log for details)

8.1.7 Version 0.6.2 (09.2018)

1. MSFragger version 20171106 was implemented
2. MODa v1.61 was implemented
3. PIPI version 1.4.5 replaced PIPI 1.3 due to compatibility issues with PIPI 1.3
4. PIPI version 1.4.6 implemented (thanks Fengchao)
5. Added latest versions of MSGF+ (version 2018.06.28 and 2018.09.12) and new python mzid to csv converte msgfplus2csv_py_v1_0_0

6. Macot wrapper and converter was implemented to allow downstream processing of Mascot results
7. Some minor bugfixes and added functionality (check git log for details)

8.1.8 Version 0.6.1 (03.2018)

1. Improved compaitibility to pymzML generation 2 and splitted up the mzML to mgf converter in version 1.0.0 (pymzML 0.7.9) and 2.0.0 (pymzML 2.0.2). The version of installed pymzML is now automatically determined and the corresponding converter version is used.
2. Added latest versions of MSGF+ (version 2018.01.20) and the corresponding C based mzidentML converter (version 1.2.0 and 1.2.1)
3. Several new engine versions were made avaliable via the install_resources script.
4. Some general code cleanup was done

8.1.9 Version 0.6.0 (01.2018)

1. Restructuring of engine classes. SEARCH_ENGINE(s) are now devided into * CROSS_LINK_SEARCH_ENGINE(s) * DE_NOVO_SEARCH_ENGINE(s) * PROTEIN_DATABASE_SEARCH_ENGINE(s) * SPECTRAL_LIBRARY_SEARCH_ENGINE(s) The META_INFO of corresponding engines has been changed accordingly. Furthermore, CONVERTER(s) have been split into CONVERTER(s) and MISC_ENGINE(s)
2. Restructuring of UController functions. Unified functions for all engines of one engine class (Converter, Search_Engines, Validation_Engines, etc) are now available. Function names for each engine class are stored in ukb.ENGINE_TYPES and ukb.UCONTROLLER_FUNCTIONS
3. ursgal_kb.py has been renamed to ukb.py
4. Implemented the following open modification search engines (as protein database search engines): MSFragger, PIPI, ModA
5. Smaller fixes and improvements (please check git log)

8.1.10 Version 0.5.0 (04.2017)

1. New branch: upapa_v3. This branch will soon be merged into the master after rigid testing and evaluation.
2. New improved peptide mapper version in terms of RAM usage and speed. Peptide mapping is now a standalone unode. Classes for mapping can be imported from anywhere from the undoe. Input for standalone node is a not-unified csv file. Branch: upapa_v3
3. Unify csv is now placed after the upeptide_mapper node if a database search engine (e.g. OMSSA, X!Tandem etc.) is used. Branch: upapa_v3
4. Unify csv was adjusted to meet the new requirements of the separated peptide mapping node. Please also note, that the default behaviour of remapping amino acid 'U' to 'C' is not longer performed.
5. Unify csv now reports if the peptide fulfills the enzyme cleavage parameters, like number of missed cleavages and if the C and N terminus is correct. Column name: 'Complies search criteria'
6. Test script update
7. Documentation update
8. Implementation of a customizable SVM for PSM post-processing

9. Smaller fixes and improvements (please check git log)

8.1.11 Version 0.4.0rc1 (05.2016)

1. included a upeptide_mapper for fast peptide to sequence mapping.
2. renamed engine folder to wrappers
3. combined all files from the kb folder into one single file, **uparams.py** which is parsed during unode initialization. Advantage is to see all params grouped together.
4. Added more information to the unique parameters, such as description and default value types.
5. Included script to auto-generate documentation from uparams file.
6. Updated documentation to reflect the changes above.

8.1.12 Version 0.3.4 (02.2016)

1. Implementation of de novo search engines: Novor, PepNovo
2. X!Tandem version Vengeance included

CHAPTER 9

Contribution Guidelines

9.1 Contribution Guidelines

Ursgal - Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis

9.1.1 Summary

In general, contribution to Ursgal is very welcome! Feel free to fork and/or clone Ursgal. If you want to improve code or contribute new nodes/tools/algorithms please read these guidelines first. If something is unclear please contact one of the authors for help or let us know via e.g. an issue.

We are happy to include your name to the list of contributors in the README. Drop a line to one of the developers if you want to get included (and of course you actually contributed something)

9.1.2 Commit messages

First of all, please be concise and as descriptive (explicit is better than implicit :)) as possible. It is always helpful to point out, which parts of Ursgal were changed/fixed (e.g. documentation or example scripts etc.). In the same time, please avoid unneccesarily long messages.

9.1.3 Parameters

The central idea of Ursgal are the unified parameters. The central parameter is tranlated, so that every engine can use it. This means, if you implement a new engine, you have to go through the (more or less) tedious process to check, if parameter X of the new engine Y is already listed in uparams.py. We require to be very thorough in this process. Having the same parameter multiple times must be avoided! There may be difficult cases, to decide if the parameter is actually the same, but by using the translation system in Ursgal, some adjusments can be made. Please refer to the documentation for further instructions and considerations on the parameters.

9.1.4 Code standards and conventions

Since this a collaborative project, you will encounter different coding styles. Despite the fact that we know that diversity is beautiful, we need to keep some common line on how to code (This list may be further extended). We generally use PEP8 style (https://www.python.org/dev/peps/pep-0008/) with the exception of E203 (whitespaces before : in order to align values in dicts). Additionally this list will give you some things to think about:

Re-think naming of variables at least twice

Re-check deleting of own debug code before sending Pull requests

Re-check own files created by nosetests and add it into '.gitignore' before sending Pull requests

9.1.5 Test philosophy

Test your code! Seriously, test you code! If you add new functionality or nodes at the same time provide (a) test function(s). We have already a set of tests and different files, which can be used for the test. Avoid adding new test files if possible to keep the repo small.

9.1.6 Sphinx guide

We use Sphinx to automatically build and format the documentation. Please keep this style in your docstrings

9.1.7 Other rules and cosiderations

None so far.

9.1.8 Merge/pull requests

Please use the pull request to push your code to the master repository. It will be automatically tested by Travis and AppVeyor if the module is still working in unix and Windows environments. Pull requests will be discussed by the main dev team and merged into Ursgal.

9.1.9 Issues

If you have an issue or problem, please first search all open issues and pull request to avoid duplication of efforts. If you have a fix for the problem you may directly open a pull requets. On the other hand, if you plan to or are already working on implementing new stuff, you may also open an issue and (pre-) announce your contribution. Please tag then the issue with 'enhancement'. In general the core team of Ursgal will also take care of crucial bugs in the main code. Since Ursgal is open source, we cannot maintain every detail and assure its compatibility and functionality (please be reminded here to test your code, seriously, test your code)

9.1.10 Citation

Be reminded, that in an academic world, citations are the only credit that one can hope for ;) Therefore, please make sure to properly cite every tool that you use or implement. And of course, if you use Ursagl, do not forget to cite us

Kremer, L. P. M., Leufken, J., Oyunchimeg, P., Schulze, S. and Fufezan, C. (2015): Ursgal, Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis , Journal of Proteome research, 15, 788-. DOI:10.1021/acs.jproteome.5b00860

CHAPTER 10

FAQ

10.1 Frequently Asked Questions

10.1.1 Installation

Q: MS Amanda does not work on Unix. What could be the problem?

To run MSAmanda one needs to install the Mono frameweork. Visit http://www.mono-project.com/ for proper installation instructions.

Q: MS-GF+ (or any Java based engine) fails. What could be the problem?

To run Java based engines like MS-GF+ or MSFragger, Java Runtime Environment needs to be installed. Visit http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html for download and installation.

Q: Downloading http files is not working on OSX. Why?

Make sure that certificates are properly installed. Go to Applications/Python 3.6 and double-click Install Certificates.command. The latest version of Python3.6 for Mac should come with the right certifications for secure connections anyways.

Q: I have problems installing pyahocorasick on Windows! What can I do?

Generally, Python 3.6 should be used when working with Windows. Here are some general remarks for a flawless installation under Windows.

When using Windows 7, additionally install: Microsoft Visual C++ Build Tools, https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2017)

When using Windows 10, consider additionally installing MS Build Tools 2015

10.1.2 Usage

Q: Found mismatch between json parameter ....

```
Found mismatch between json parameter csv_filter_rules:
[['PEP', 'lte', 0.01], ['Is decoy', 'equals', 'false']] and
controller params csv_filter_rules:
[('PEP', 'lte', 0.01), ('Is decoy', 'equals', 'false')].
Consider re-run with force=True or delete old u.jsons.
```

During JSON dump Python tuples are converted into list like objects, thus this might be a reason. Just change your parameter to lists instead of tuples :)

Q: How do I add an engine that is not installed via install_resources.py?

Download the engine from the respective developers homepage (links are given in the wrapper documentation of the respective engine). Create a folder in the corresponding Ursgal resoucres (name of the folder = name of the engine in Ursgal, for more information see *Create/Implement your own UNode*: 1.Integration into Resources) and unpack/save all required files, especially the executable, there. Remember to run:

```
user@localhost:~/ursgal$ python3.4 setup.py install
```

to include Ursgal (and the changes you have made to the resources) into Python site-packages.

Q: The example script simple_example_search.py fails. What am I doing wrong?

Check the printouts: at which step is it failing? If the download of the example BSA1.mzML was not successful, and you're using OSX, see *Q: Downloading http files is not working on OSX. Why?*. If MS-GF+ fails and you are not sure if you have installed Java Runtime Environment, see *Q: MS-GF+ (or any Java based engine) fails. What could be the problem?*. If this doesn't help, shoot us a message or open an issue on GitHub (please include your printouts).

Q: A validation engine (Percolator, qvality, ...) fails. What's going on?

There are two common problems causing your workflow to fail at the point of validating results: 1. Your database doesn't contain decoys (check out *target_decoy_generation_example.py*) or decoys are not recognized (check if the uparam 'decoy_tag' is correct for your database). 2. Your list of results is too small for proper statistics (the error message is something like "Too good seperation between targets and decoys"). In this case, you need to improve your search parameters (e.g. mass tolerances), database size (e.g. whole proteome instead of a single protein) or MS measurements (i.e. your raw data).

Q: An engine fails with a certain combination of parameter values. Why is that not checked beforehand?

In general, we don't check if a vertain combination of parameter values is allowed or makes sense, since this would be quite some work for all the different engines and use cases. Please check the documentation of the respective engines to avoid these issues.

Examples:

- MS-GF+ does not allow to specify a maximum number of missed cleavages wgen using unspecific cleavage -> use params['max_missed_cleavages'] = -1

10.1.3 Development

Q: How do I create/add a new engine?

See *Create/Implement your own UNode*.

Q: How do I keep Ursgal up-to-date?

Ursgal is still in development and changes, extensions, etc. are pushed to GitHub. Therefore, the easiest way (if you have cloned Ursgal from GitHub) is:

```
user@localhost:~/ursgal$ git pull
```

If you have not cloned Ursgal but used the ZIP file you can replace the folder with the newly downloaded and extracted version.

In both cases you might need to run the setup again to update the python site-packages:

```
user@localhost:~/ursgal$ python3 setup.py install
```

CHAPTER 11

Known Issues

11.1 Known Issues

11.1.1 General

- Java used memory size

  Ajust the memory usage by Java according to your needs. When using memory intensive tasks as mzIdentML conversion of large files, an adjustement of the Java Xmx values may be required. The default is the usage of 13 GB of your RAM. Please refer to the Java documentation for further information. http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/java.html In Ursgal the parameter *java_-Xmx* can be used to adjust the Java memory usage.

- MS-GF+ (or another Java based engine) crashes

  Please make sure that you have installed the current Java Runtime Environment Download (Java SE Development Kit 8u131):

  ```
  http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-
  ↪2133151.html
  ```

11.1.2 Windows general

- Some modules can not be compiled/installed with python3.4+ using pip.

  E.g. pyahocorasick can not be installed. This has the consequence, that on Windows the old peptide mapper version is used. There are several workarounds to compile and install the modules manually, e.g.:

  http://haypo-notes.readthedocs.io/python.html#build-a-python-wheel-package-on-windows

11.1.3 Windows 10

- MS Amanda can not load .fasta files
- **calculating the md5 can cause problems e.g. while executing test.** This is due to different line endings on Unix and Windows systems. The test functions test for both md5, so this problem should be avoided.

11.1.4 MONO

- **Mono is the .NET replacement under *nix systems, since .NET is not directly** ported by Microsoft to other systems than windows. Unfortunately mono is not as stable as the official .NET build. Therefore:

  MS Amanda crashes randomly under *nix systems (e.g. Linux or OS X)

Python Module Index

u

Index

Symbols

A

B

C

D

E

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

V

W

X