ursgal Documentation

Release ursgal_version = '0.5.0'

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

Aug 11, 2017

Contents

The latest Documentation was generated on: Aug 11, 2017

CHAPTER 1

Introduction

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

Summary

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

1. Peptide spectrum matching with up to five different search engines (some available in multiple versions)
2. Evaluation and post processing of search results with up to two different engines
3. Integration of search results from different search engines
4. Creation of a target decoy database

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

Download

Get the latest version via GitHub:

https://github.com/ursgal/ursgal

as .zip package:

https://github.com/ursgal/ursgal/archive/master.zip

or via git clone URL:

https://github.com/ursgal/ursgal.git

The complete Documentation can be found at

http://ursgal.readthedocs.org/

Installation

Ursgal requires Python 3.4 or higher.

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

- GitHub version: Starting with this 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

Install requirements:

```
user@localhost:~$ cd ursgal
user@localhost:~/ursgal$ pip3.4 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

Install third party engines:

```
user@localhost:~/ursgal$ python3.4 install_resources.py
```

Install Ursgal:

```
user@localhost:~/ursgal$ python3.4 setup.py install
```

Note: Under Linux, it may be required to change the permission in the python3.4 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.)

Tests

Run nosetests in root folder. You might need to install nose for Python3 first although it is in the requirements.txt (above) thus pip3.4 install -r requirements should have installed it already. Then just execute:

```
user@localhost:~/ursgal$ nosetests3
```

to test the package.

Participate

Fork us at https://github.com/ursgal/ursgal and open up pull requests! Thanks!

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

Documentation

For more detailed documentation of the modules and examples, please refer to the documentation folder or http://ursgal.readthedocs.org

Disclaimer

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

Copyrights

- Lukas Vaut

Contact

Dr. Christian Fufezan
Institute of Plant Biology and Biotechnology
Schlossplatz 8 , R 105
University of Muenster
Germany
eMail: christian@fufezan.net
Tel: +049 251 83 24861

http://www.uni-muenster.de/Biologie.IBBP.AGFufezan

Citation

Ursgal citation

Lukas P. M. Kremer, Johannes Leufken, Purevdulam Oyunchimeg, Stefan Schulze, and Christian Fufezan (2015): Ursgal, Universal Python Module Combining Common Bottom-Up Proteomics Tools for Large-Scale Analysis, Journal of Proteome research, DOI:10.1021/acs.jproteome.5b00860

Note: Please cite every tool you use in Ursgal. During runtime the references of the tools you were 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.
- 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.
- Ma, B. Novor: real-time peptide de novo sequencing software. J Am Soc Mass Spectrom. 2015 Nov;26(11):1885-94

- 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.

CHAPTER 2

Quick Start

Quick Start Tutorial

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

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. 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*).

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+'
)
```

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.add_estimated_fdr()*** Estimates the false discovery rate of target-decoy-based result files, given a quality score
- ***UController.combine_search_results()*** Statistical integration of search results from multiple engines
- ***UController.fetch_file()*** Downloads files (HTTP or FTP)
- ***UController.filter_csv()*** Filters csv files row-wise according to custom filtering rules
- ***UController.generate_target_decoy()*** Generates a target-decoy database from a regular fasta database
- ***UController.merge_csvs()*** Merges csv files
- ***UController.search()*** Peptide spectrum matching
- ***UController.validate()*** Statistical post-processing of search results
- ***UController.visualize()*** Visualization of results

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_piledriver', 'msgfplus',
                  'msamanda_1_0_0_5243', '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_piledriver_unified.csv', 'fileB_xtandem_piledriver_unified.csv',
```

```
 'fileA_msgfplus_v9979_unified.csv', 'fileB_msgfplus_v9979_unified.csv',
 'fileA_msamanda_1_0_0_5243_unified.csv', 'fileB_msamanda_1_0_0_5243_unified.csv',
 'fileA_myrimatch_2_1_138_unified.csv', 'fileB_myrimatch_2_1_138_unified.csv']
```

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'
)
```

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

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).

Schematic Overview

Resources

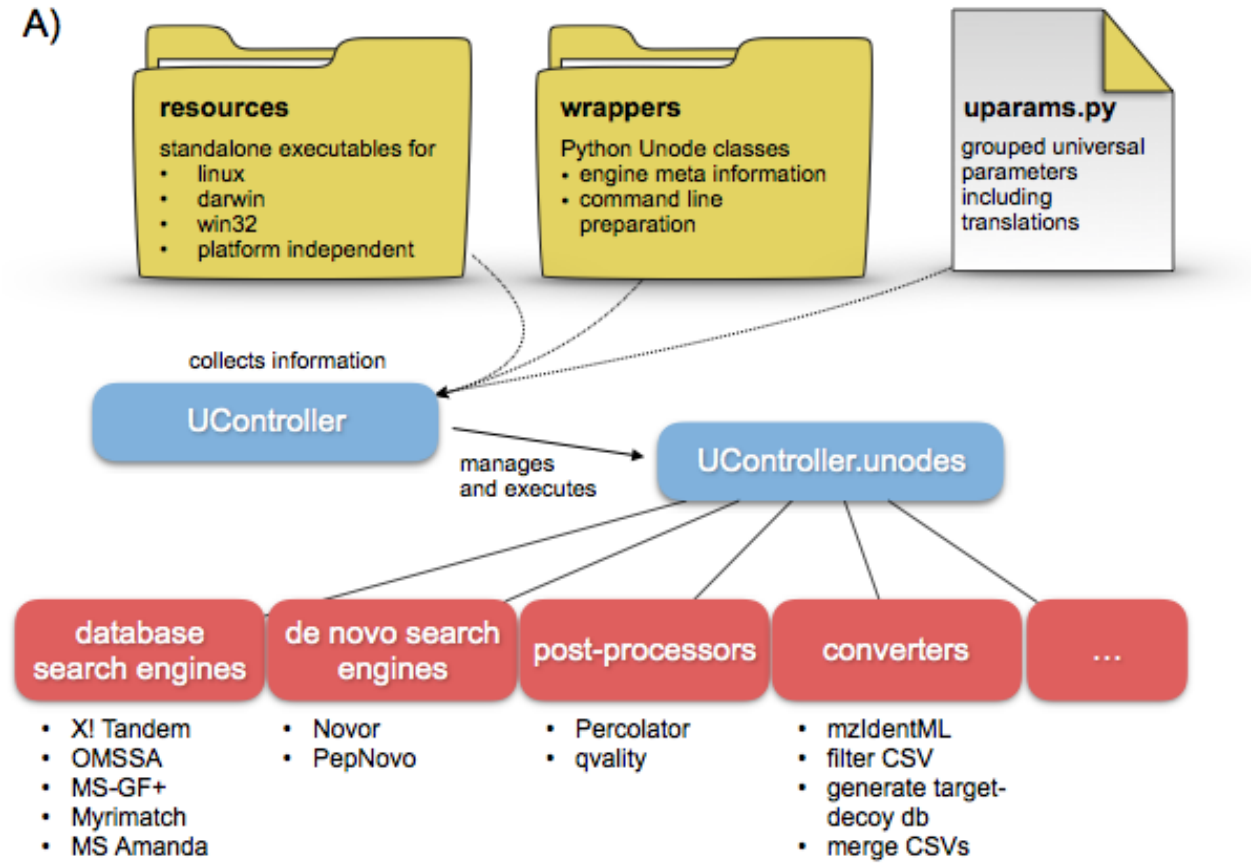
The **resources/** directory contains the main code for each UNode, e.g.:

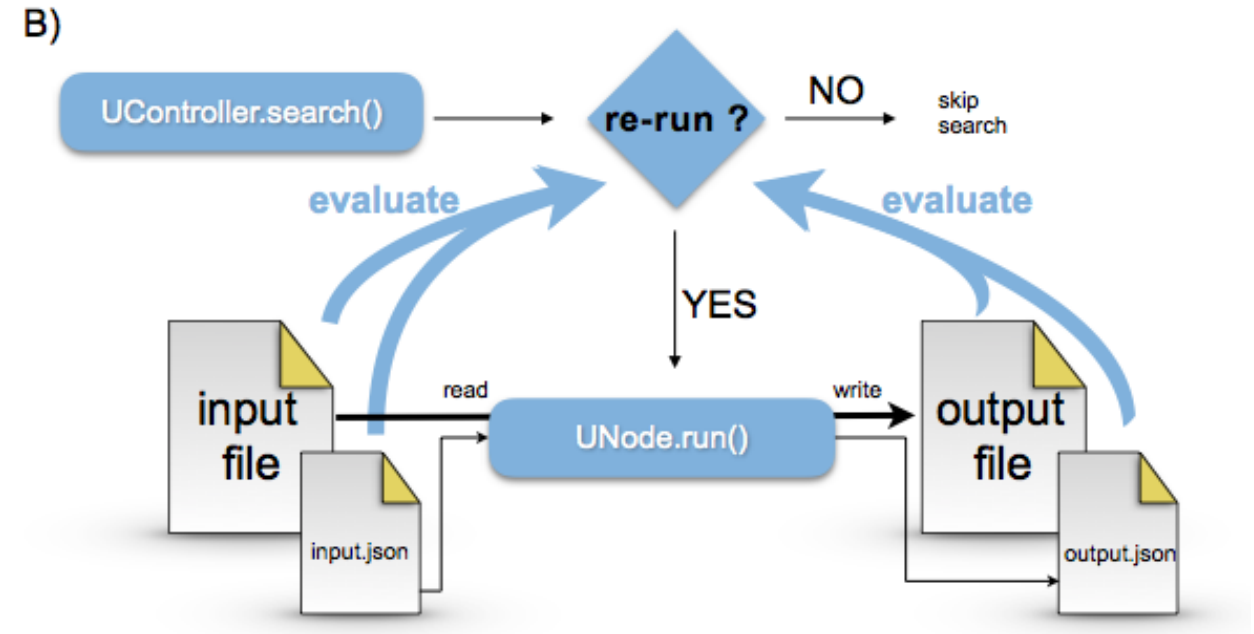
1. executables (i.e. .exe or .jar)
2. standalone Python scripts
3. any additional files that are required by the engine

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

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 python3.4
import ursgal

class omssa_2_1_9( ursgal.UNode ):
    """
    omssa_2_1_9 UNode

    Parameter options at http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/asn_spec/
↪omssa.asn.html

    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.

    """
    META_INFO = { ... }

    def __init__(self, *args, **kwargs):
        super(omssa_2_1_9, 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.

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.

```
META_INFO = {
    'engine_type'             : {
      'controller'         : False,
      'converter'          : False,
      'validation_engine'  : False,
      'search_engine'      : True,
      'meta_engine'        : 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' : 'Geer LY, Markey SP, Kowalak JA, '\
        'Wagner L, Xu M, Maynard DM, Yang X, Shi W, Bryant SH (2004) '\
        'Open Mass Spectrometry Search Algorithm.',
    ...
}
```

Input types

Input types are currently not used but the next iteration will include this. For example.

```
META_INFO = {
    ...
    'input_types'             : ['.mgf'],
    ...
}
```

Output Extension

The output extension is required to auto-generate the output file name. For example:

```
META_INFO = {
    ...
    'output_extension'        : '.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,
    ...
}
```

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',
                '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'],
            },
        },
    },
    ...
}
```

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.

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.
- **'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 information for possible parameter value ranges and step sizes.

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

```
ursgal_params = {
    'frag_mass_tolerance' : {
            'available_in_unode' : [
                'msamanda_1_0_0_5242',
                'msamanda_1_0_0_5243',
                'myrimatch_2_1_138',
                'myrimatch_2_2_140',
                'novor_1_1beta',
                'omssa_2_1_9',
                'pepnovo_3_1',
                'xtandem_cyclone_2010',
                'xtandem_jackhammer',
                'xtandem_piledriver',
                'xtandem_sledgehammer',
                'xtandem_vengeance',
            ],
            'default_value' : 20,
            'description' :  ''' Mass tolerance of measured and calculated fragment␣
↪ions ''',
            'triggers_rerun' : True,
            'ukey_translation' : {
                'msamanda_style_1' : 'ms2_tol',
                'myrimatch_style_1' : 'FragmentMzTolerance',
                'novor_style_1' : 'fragmentIonErrorTol',
                'omssa_style_1' : '-to',
                'pepnovo_style_1' : '-fragment_tolerance',
                'xtandem_style_1' : 'spectrum, fragment monoisotopic mass error',
            },
            'utag' : [
                'fragment',
            ],
            'uvalue_translation' : {
            },
            'uvalue_type' : "int",
            'uvalue_option' : {
                'min': 0,         # default = |default_value * 100| * -1
                'max': 100000,    # default = |default_value * 100|
                'updownval': 1,   # default = 1
        },
    },
    ...
```

```
}
```

CHAPTER 4

Ursgal module contents

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/kb/*.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' }
...)
```

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

The UController add_estimated_fdr function

Parses a target/decoy search result file and adds a column called "estimated_FDR".

The CSV must contain:

•a column with a quality score for each PSM (e-value, error probability etc.)

•a column called "Is decoy" indicating whether a PSM is decoy or target.

Keyword Arguments

- **input_file** (*str*) – The complete path to the input, input file has to be a .csv file meeting the criteria described above.
- **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.params['validation_score_field'] = 'e-value'
>>> uc.params['bigger_scores_better']   = False
>>> uc.add_estimated_fdr(
...     input_file = 'my_search_results.csv',
... )
```

Note: This function can be used to independently compare the performance of different quality scores (where performance is the ability to distinguish target PSMs from decoy PSMs).

Returns Path of the output file

Return type str

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

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.

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

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

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. Is 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_unode(*input_file*, *engine*, *force=False*, *output_file_name=None*, *dry_run=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*)

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*)
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*)

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*)
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

merge_fdicts(**fdicts*)

prepare_resources(*root_zip_target_folder*)

run_unode_if_required(*force*, *engine_name*, *answer*, *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_csv(*input_file*, *force=False*, *output_file_name=None*)
The UController sanitize_csv function

Result files (.csv) are sanitized following defined parameters. That means, for each spectrum PSMs are compared and the best spectrum (spectra) is (are) chosen.

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 parameters have to be defined in the params. See the engine documentation for further information (*sanitize_csv_1_0_0._execute()*).

Example

```
>>> # Only the best PSM for one spectrum is retained
>>> # and only if its PEP is differing from the secondbest by
>>> # two orders of magnitude
>>> uc.params['validation_score_field'] = 'PEP'
>>> uc.params['bigger_scores_better'] = False
>>> uc.params['score_diff_threshold'] = 2
>>> uc.params['threshold_is_log10'] = True
>>> uc.sanitize_csv( 'my_results.csv' )
```

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*, *force=None*, *output_file_name=None*)
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_to_mgf_and_update_rt_lookup()* (if required)
- *search_mgf()*
- *convert_results_to_csv()*
- *map_peptides_to_fasta()*
- *unify_csv()*

---

search_mgf(*input_file*, *engine*, *force=None*, *output_file_name=None*)

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',
...     '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*)
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*, *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*, *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

UNode

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

ursgal class

__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 '_'

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 = ["search_engine"]
    )
>>> print( last_engine )
"xtandem_sledgehammer"
```

Returns

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

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']

UCore

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_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.**digest**(*sequence*, *enzyme*, *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
- **no_missed_cleavages** (*bool*) – allow missed cleavages or not

Returns list of digestes peptides

Return type list

ursgal.ucore.**parseFasta**(*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

UMapMaster

Mapping classes of Ursgal

UParamMapper

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

UParamMapper class offers interface to ursgal.uparams

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

group_styles()

Parses self.items() and build up lookups. Additionally, consistency check is performed to guarantee that each engine is mapping only on one style.

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*)

yields all mapping dicts

Chemical Composition

class ursgal.**ChemicalComposition**(*sequence=None*, *aa_compositions=None*, *isotopic_distributions=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*)
Adds chemical formula to the instance

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
dict – 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.

---

composition_of_aa_at_pos = None
dict – 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.

---

composition_of_mod_at_pos = None
dict – 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.

---

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*)
Subtract chemical formula from instance

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*) –

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/kb/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*)–

CHAPTER 5

Engines

Ursgal allows existing programs to be incorporated with ease. We call those programs or scripts engines. Currently, ursgal comes with these engines:

Included engines

Available Search Engines

MS-GF+

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

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.

preflight()

Formatting the command line via self.params

Modifications file will be created in the output folder

Returns self.params

Return type dict

X!Tandem

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

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

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

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*)

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)

MSFragger

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. Nat. Publ. Gr. 293

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 paramn input file via self.params

Returns self.params

Return type dict

MS Amanda

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

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)

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

Available De Novo Engines

Novor

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

Available Cross Linking 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

Other 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

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* *output_f* *score* *score*

Convert csvs to ssl

Convert MS-GF+ MZID to CSV v2016_09_16

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 1_0_0

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

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

_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_

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

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

MzidLib 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

MzidLib 1_6_11

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

Percolator 2_08

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

Plot pyGCluster heatmap from CSV 1_0_0

class ursgal.wrappers.plot_pygcluster_heatmap_from_csv_1_0_0.**plot_pygcluster_heatmap_from_csv**

plot_pygcluster_heatmap_from_csv_1_0_0 UNode

_execute()

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

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

_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

Unify CSV v1_0_0

class ursgal.wrappers.unify_csv_1_0_0.**unify_csv_1_0_0**(**args*, ***kwargs*)
unify_csv_1_0_0 UNode

_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.

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

_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_ma
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'
    }
]
```

Mapper class v3 (dev)

class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_ma
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

Mapper class v2 (deprecated)

class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_ma
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

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

results for the different areas e.g. dict['C-(A|B|D)']['results']

Output file is written to the common_top_level_dir

Return type dict

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 ...

How to extend and create new engines ?

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}',
            '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.

Example add write_unfiltered_results for filter_csv_1_0_0

```
'write_unfiltered_results' : {
        'available_in_unode' : [
            'filter_csv_1_0_0',
        ],
        'default_value' : False,
        'description' :  ''' writes rejected results if True ''',
        'trigger_rerun' : True,
        'ukey_translation' : {
            'filter_csv_style_1' : 'write_unfiltered_results',
        },
        'utag' : [
            'conversion',
        ],
        'uvalue_translation' : {
        },
        'uvalue_type' : "bool",
    },
```

After changing uparams.py, please run rewrite_uparams.py to check for duplicates/errors and to sort the dictionary.

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']),
        '--another_parameter',
        '{another_param}'.format(**self.params['translations']),

    ]
```

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):
    convert_xtandemXML_to_identcsv( self.params )
```

Example needs to be exchanged

Example: ursgal/engines/msgfplus_v9979.py

```
#!/usr/bin/env python3.4
import ursgal
import os

class msgfplus_v9979( ursgal.UNode ):
    """
    MSGF+ UNode
    Parameter options at https://bix-lab.ucsd.edu/pages/viewpage.action?
↪pageId=13533355
    """
    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
        '''

        self.params['mgf_input_file'] = os.path.join(
            self.params['input_dir_path'],
            self.params['file_root'] + '.mgf'
        )

        self.params['output_file_incl_path'] = os.path.join(
            self.params['output_dir_path'],
            self.params['output_file']
        )

        self.params['modification_file'] = os.path.join(
            self.params['output_dir_path'],
            self.params['output_file'] + '_Mods.txt'
        )
        self.created_tmp_files.append( self.params['modification_file'] )

        mods_file = open( self.params['modification_file'], 'w', encoding = 'UTF-8' )
        modifications = []
        cam = False
        for t in [ 'fix', 'opt' ]:
            for mod in self.params[ 'mods' ][ t ]:
                if self.params['label'] == '15N' and mod[ 'aa' ] == 'C' and mod[ 'name
↪' ] == 'Carbamidomethyl':
                    cam = True
                    continue
                modifications.append( '{0},{1},{2},{3},{4}'.format(mod[ 'mass' ],␣
↪mod[ 'aa' ], t, mod[ 'pos' ], mod[ 'name' ] ) )
```

```
        if self.params['label'] == '15N':
            DICT_15N_DIFF = ursgal.kb.ursgal.DICT_15N_DIFF
            for aa, mass in DICT_15N_DIFF.items():
                if cam:
                    if aa == 'C':
                        modifications.append( '{0},{1},fix,any,15N {2}'.format(␣
↪float(mass) + 57.021464, aa, aa + 'Carbamidomethyl') )
                    else:
                        modifications.append( '{0},{1},fix,any,15N {1}'.format( mass,␣
↪aa ) )
                else:
                    modifications.append( '{0},{1},fix,any,15N {1}'.format( mass, aa␣
↪) )

        for mod in modifications:
            print( mod, file = mods_file )
        mods_file.close()

        self.params[ 'command_list' ] = [
            'java', '-Xmx{java_-Xmx}'.format( **self.params), '-jar', self.exe,
            '-s', '{mgf_input_file}'.format( **self.params),
            '-d', '{database}'.format(**self.params),
            '-o', '{output_file_incl_path}'.format(**self.params),
            '-t', '{precursor_mass_tolerance_minus}{precursor_mass_tolerance_unit},
↪{precursor_mass_tolerance_plus}{precursor_mass_tolerance_unit}'.format(**self.
↪params),
            '-ti', '{precursor_isotope_range}'.format(**self.params),
            '-thread', '{cpus}'.format(**self.params),
            '-tda', '0',
            '-m', '{frag_method}'.format(**self.params),
            '-inst', '{instrument}'.format(**self.params),
            '-e', '{enzyme}'.format(**self.params),
            '-ntt', '{semi_enzyme}'.format(**self.params),
            '-mod', '{modification_file}'.format(**self.params),
            '-minLength', '{min_pep_length}'.format(**self.params),
            '-maxLength', '{max_pep_length}'.format(**self.params),
            '-minCharge', '{precursor_min_charge}'.format(**self.params),
            '-maxCharge', '{precursor_max_charge}'.format(**self.params),
            '-n', '{num_match_spec}'.format(**self.params),
            '-addFeatures', '1',
        ]

        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 python3.4
import ursgal
import importlib
import os
import sys
import pickle
import shutil

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,
            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 6

Parameter

Ursgal to Engine Parameters overview

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

-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_v9979
- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

- msfragger_20170103

Ursgal value translations for *-xmx*

Style	Translation
msfragger_style_1	-Xmx
msgfplus_style_1	-Xmx
mzidentml_style_1	-Xmx

_extentions

information of extentions

Default value ['.csv', '.den', '.dta', '.dta.txt', '.fasta', '.gaml', '.kojak.txt', '.mgf', '.ms2', '.mzData', '.mzML', '.mzML.gz', '.mzXML', '.mzid', '.mzid.gz', '.pep.xml', '.perc.inter.txt', '.perc.intra.txt', '.perc.loop.txt', '.perc.single.txt', '.pkl', '.raw', '.svg', '.tsv', '.txt', '.xml', '.xml.gz']

type dict

triggers rerun False

Available in unodes

Ursgal value translations for *_extentions*

Style	Translation

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)

Default value ['J', 'O']

type dict

triggers rerun False

Available in unodes

- unify_csv_1_0_0
- upeptide_mapper_1_0_0
- compomics_utilities_4_11_5

Ursgal value translations for *aa_exception_dict*

Style	Translation
compomics_utilities_style_1	aa_exception_dict
unify_csv_style_1	aa_exception_dict
upeptide_mapper_style_1	aa_exception_dict

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 False

Available in unodes

- sanitize_csv_1_0_0

Ursgal value translations for *accept_conflicting_psms*

Style	Translation
sanitize_csv_style_1	accept_conflicting_psms

add_cterm_peptide

Statically add mass in Da to C-terminal of peptide

Default value 0.0

type float

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *add_cterm_peptide*

Style	Translation
msfragger_style_1	add_Cterm_peptide

add_nterm_peptide

Statically add mass in Da to N-terminal of peptide

Default value 0.0

type float

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *add_nterm_peptide*

Style	Translation
msfragger_style_1	add_Nterm_peptide

allow_multiple_variable_mods_on_residue

static mods are not considered

Default value 1

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *allow_multiple_variable_mods_on_residue*

Style	Translation
msfragger_style_1	allow_multiple_variable_mods_on_residue

base_mz

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

Default value 1000

type int

triggers rerun False

Available in unodes

- moda_v1_51
- omssa_2_1_9
- pepnovo_3_1

Ursgal value translations for *base_mz*

Style	Translation
moda_style_1	base_mz
omssa_style_1	base_mz
pepnovo_style_1	base_mz

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

Ursgal value translations for *batch_size*

Style	Translation
myrimatch_style_1	NumBatches
xtandem_style_1	spectrum, sequence batch size

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 False

Available in unodes

- add_estimated_fdr_1_0_0
- percolator_2_08
- qvality_2_02
- sanitize_csv_1_0_0
- svm_1_0_0

Ursgal value translations for *bigger_scores_better*

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

Ursgal key translations

Ursgal Value	Translated Value				
.	add_estimated_fdr_spectrum	calculator_style_1	qvality_style_1	sanitize_csv_style_1	svm_style_1
None	None	None	None	None	None
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
msfragger_20170103	1	1	1	1	1
msgfplus_v2016_09_16	0	0	0	0	0
msgfplus_v2017_01_27	0	0	0	0	0
msgfplus_v9979	0	0	0	0	0
myrimatch_2_1_138	1	1	1	1	1
myrimatch_2_2_140	1	1	1	1	1
omssa_2_1_9	0	0	0	0	0
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

cleavage_cterm_mass_change

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

Default value 17.00305

type float

triggers rerun False

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer

- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *cleavage_cterm_mass_change*

Style	Translation
xtandem_style_1	protein, cleavage C-terminal mass change

cleavage_nterm_mass_change

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

Default value 1.00794

type float

triggers rerun False

Available in unodes

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

Ursgal value translations for *cleavage_nterm_mass_change*

Style	Translation
xtandem_style_1	protein, cleavage N-terminal mass change

clip_nterm_m

Specifies the trimming of a protein N-terminal methionine as a variable modification (0 or 1)

Default value 0

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *clip_nterm_m*

Style	Translation
msfragger_style_1	clip_nTerm_M

compensate_small_fasta

Compensate for very small database files.

Default value False

type bool

triggers rerun False

Available in unodes

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

Ursgal value translations for *compensate_small_fasta*

Style	Translation
xtandem_style_1	scoring, cyclic permutation

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

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 False

Available in unodes

- compomics_utilities_4_11_5

Ursgal value translations for *compomics_utility_name*

Style	Translation
compomics_utilities_style_1	compomics_utility_name

compomics_version

Defines the compomics version to use

Default value compomics_utilities_4_11_5

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *compomics_version*

Style	Translation
ucontroller_style_1	compomics_version

compress_raw_search_results_if_possible

Compress raw search result to .gz: True or False

Default value True

type bool

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *compress_raw_search_results_if_possible*

Style	Translation
ucontroller_style_1	compress_raw_search_results_if_possible

compute_xcorr

Compute xcorr

Default value False

type bool

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *compute_xcorr*

Style	Translation
myrimatch_style_1	ComputeXCorr

Ursgal key translations

Ursgal Value	Translated Value
.	myrimatch_style_1
0	0
1	1

consecutive_ion_prob

Probability of consecutive ion (used in correlation correction)

Default value 0.5

type float

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *consecutive_ion_prob*

Style	Translation
omssa_style_1	-scorp

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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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

Ursgal value translations for *cpus*

Style	Translation
kojak_style_1	cpus
msfragger_style_1	num_threads
msgfplus_style_1	-thread
myrimatch_style_1	-cpus
omssa_style_1	-nt
ucontroller_style_1	cpus
xtandem_style_1	spectrum, threads

Ursgal key translations

Ursgal Value	Translated Value					
.	kojak_style_1	msgfplus_style_1	myrimatch_style_1	omssa_style_1	ucontroller_style_1	xtandem_style_1
-1	max - 1	max - 1	max - 1	max - 1	max - 1	max - 1

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *cross_link_definition*

Style	Translation
kojak_style_1	cross_link_definition

csv_filter_rules

Rules are defined as list of tuples with the first tuple element as the column name/csv fieldname, the second tuple element the rule and the third tuple element the value which should be compared

Default value None

type list

triggers rerun False

Available in unodes

- filter_csv_1_0_0

Ursgal value translations for *csv_filter_rules*

Style	Translation
filter_csv_style_1	filter_rules

Ursgal key translations

Ursgal Value	Translated Value
.	filter_csv_style_1

database

Path to database file containing protein sequences in fasta format '' : None

Default value None

type str

triggers rerun False

Available in unodes

- kojak_1_5_3
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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

Ursgal value translations for *database*

Style	Translation
compomics_utilities_style_1	database
kojak_style_1	database
moda_style_1	Fasta
msamanda_style_1	database
msfragger_style_1	database_name
msgfplus_style_1	-d
myrimatch_style_1	ProteinDatabase
omssa_style_1	-d
unify_csv_style_1	database
upeptide_mapper_style_1	database
xtandem_style_1	file URL

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 False

Available in unodes

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

Ursgal value translations for *database_taxonomy*

Style	Translation
xtandem_style_1	taxon label

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
all	0

decoy_generation_mode

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

Default value shuffle_peptide

type select

triggers rerun False

Available in unodes

- generate_target_decoy_1_0_0

Ursgal value translations for *decoy_generation_mode*

Style	Translation
generate_target_decoy_style_1	mode

decoy_tag

decoy-specific tag to differentiate between targets and decoys

Default value decoy

type str

triggers rerun False

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

Ursgal value translations for *decoy_tag*

Style	Translation
generate_target_decoy_style_1	decoy_tag
kojak_style_1	decoy_tag
myrimatch_style_1	DecoyPrefix
mzidentml_style_1	-decoyRegex
unify_csv_style_1	decoy_tag
upeptide_mapper_style_1	decoy_tag
xtandem2csv_style_1	decoy_tag

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 False

Available in unodes

- ucontroller

Ursgal value translations for *del_from_params_before_json_dump*

Style	Translation
ucontroller_style_1	del_from_params_before_json_dump

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 False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *denovo_model*

Style	Translation
pepnovo_style_1	-model

Ursgal key translations

Ursgal Value	Translated Value
.	pepnovo_style_1
cid_trypsin	CID_IT_TRYP

denovo_model_dir

Directory containing the model files for PepNovo. If 'None', it is supposed to be in :
resources/<platform>/<architecture>/pepnovo_3_1

'' : None

Default value None

type str

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *denovo_model_dir*

Style	Translation
pepnovo_style_1	-model_dir

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 False

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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *engine_internal_decoy_generation*

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

Ursgal key translations

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

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 False

Available in unodes

- ucontroller

Ursgal value translations for *engines_create_folders*

Style	Translation
ucontroller_style_1	engines_create_folders

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 False

Available in unodes

- generate_target_decoy_1_0_0
- kojak_1_5_3
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- percolator_2_08
- unify_csv_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103

Ursgal value translations for *enzyme*

Style	Translation
generate_target_decoy_style_1	enzyme
kojak_style_1	enzyme
moda_style_1	Enzyme
msamanda_style_1	enzyme specificity
msfragger_style_1	enzyme
msgfplus_style_1	-e
myrimatch_style_1	CleavageRules
novor_style_1	enzyme
omssa_style_1	-e
pepnovo_style_1	-digest
percolator_style_1	enzyme
unify_csv_style_1	enzyme
xtandem_style_1	protein, cleavage site

Ursgal key translations

Ursgal Value	Translated Value			
.	generate_target_decoy_style_1	kojak_style_1	moda_style_1	msamanda_style_1
alpha_lp	n/t	n/t	n/t	n/t
argc	R;C;P	n/t	argc, R/C	R;after;P
aspn	D;N;	n/t	aspn, D/N;	D;before;
aspn_gluc	n/t	n/t	n/t	n/t
chymotrypsin	FMWY;C;P	n/t	chymotrypsin, FMWY/C	FMWY;after;P
chymotrypsin_p	FMWY;C;	n/t	chymotrypsin, FMWY/C	FMWY;after;
clostripain	R;C;	n/t	clostripain, R/C	R;after;
cnbr	M;C;P	n/t	cnbr, M/C	M;after;P
elastase	AGILV;C;P	n/t	elastase, AGILV/C	AGILV;after;P
formic_acid	D;C;P	n/t	formic_acid, D/C	D;after;P
gluc	DE;C;P	[DE]|{P}	gluc, DE/C	DE;after;P
gluc_bicarb	E;C;P	n/t	gluc_bicarb, E/C	E;after;P
iodosobenzoate	W;C;	n/t	iodosobenzoate, W/C	W;after;
lysc	K;C;P	n/t	lysc, K/C	K;after;P
lysc_gluc	DEK;C;P	[DEK]|{P}	n/t	DEK;after;P
lysc_p	K;C;	n/t	lysc_p, K/C	K;after;
lysn	K;N;	|[K]	lysn, K/N	K;before;
lysn_promisc	AKRS;N;	n/t	lysn_promisc, AKRS/N	AKRS;before;
no_cleavage	n/t	n/t	NONE	n/t
nonspecific	n/t	n/t	n/t	;;
pepsina	FL;C;	n/t	pepsina, FL/C	FL;after;
protein_endopeptidase	P;C;	n/t	protein_endopeptidase, P/C	P;after;
staph_protease	E;C;	n/t	staph_protease, E/C	E;after;
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	KR;C;P	[KR]|{P}	trypsin, KR/C	KR;after;P
trypsin_chymotrypsin	n/t	n/t	n/t	n/t
trypsin_cnbr	KRM;C;P	n/t	trypsin_cnbr, KRM/C	KRM;after;P
trypsin_gluc	DEKR;C;P	n/t	trypsin_gluc, DEKR/C	DEKR;after;P
trypsin_p	KR;C;	[RK]|	trypsin_p, KR/C	KR;after;

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 False

Available in unodes

- svm_1_0_0

Ursgal value translations for *fdr_cutoff*

Style	Translation
svm_style_1	fdr_cutoff

filter_csv_converter_version

filter csv converter version: version name

Default value filter_csv_1_0_0

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *filter_csv_converter_version*

Style	Translation
ucontroller_style_1	filter_csv_converter_version

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 False

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *forbidden_cterm_mods*

Style	Translation
xtandem_style_1	residue, potential modification mass

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 False

Available in unodes

- novor_1_1beta

Ursgal value translations for *forbidden_residues*

Style	Translation
novor_style_1	forbiddenResidues

force

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

Default value False

type bool

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *force*

Style	Translation
ucontroller_style_1	force

frag_mass_tolerance

Mass tolerance of measured and calculated fragment ions

Default value 20

type int

triggers rerun False

Available in unodes

- 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
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103

Ursgal value translations for *frag_mass_tolerance*

Style	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
xtandem_style_1	spectrum, fragment monoisotopic mass error

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 False

Available in unodes

- 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
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103

Ursgal value translations for *frag_mass_tolerance_unit*

Style	Translation
moda_style_1	FragTolerance
msamanda_style_1	ms2_tol unit
msfragger_style_1	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
xtandem_style_1	spectrum, fragment monoisotopic mass error units

Ursgal key translations

Ursgal Value	Translated Value					
.	msamanda_style_1	msfrag-ger_style_1	myri-match_style_1	novor_style_1	omssa_style_1	xtan-dem_style_1
da	Da	0	Da	Da	Da	Daltons
ppm	n/t	1	n/t	n/t	n/t	n/t

frag_mass_type

Fragment mass type: monoisotopic or average

Default value monoisotopic

type select

triggers rerun False

Available in unodes

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

Ursgal value translations for *frag_mass_type*

Style	Translation
omssa_style_1	-tom
xtandem_style_1	spectrum, fragment mass type

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
average	1
monoisotopic	0

frag_max_charge

Maximum fragment ion charge to search.

Default value 4

type int

triggers rerun False

Available in unodes

- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *frag_max_charge*

Style	Translation
msfragger_style_1	max_fragment_charge
omssa_style_1	-zoh

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 False

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- novor_1_1beta

Ursgal value translations for *frag_method*

Style	Translation
msgfplus_style_1	-m
novor_style_1	fragmentation

Ursgal key translations

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

frag_min_mz

Minimal considered fragment ion m/z

Default value 150

type int

triggers rerun False

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance

- xtandem_alanine

Ursgal value translations for *frag_min_mz*

Style	Translation
xtandem_style_1	spectrum, minimum fragment mz

ftp_blocksize

Blocksize for ftp download

Default value 1024

type int

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_blocksize*

Style	Translation
get_http_style_1	ftp_blocksize

ftp_folder

ftp folder that should be downloaded '' : None

Default value None

type str

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_folder*

Style	Translation
get_http_style_1	ftp_folder

ftp_include_ext

Only files with the defined file extension are downloaded with ftp download '' : None

Default value None

type str

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_include_ext*

Style	Translation
get_http_style_1	ftp_include_ext

ftp_login

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

Default value None

type str

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_login*

Style	Translation
get_http_style_1	ftp_login

ftp_max_number_of_files

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

Default value None

type int

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_max_number_of_files*

Style	Translation
get_http_style_1	ftp_max_number_of_files

ftp_output_folder

Default ftp download path '' : None

Default value None

type str

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_output_folder*

Style	Translation
get_http_style_1	ftp_output_folder

ftp_password

ftp download password '' : None

Default value None

type str_password

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_password*

Style	Translation
get_http_style_1	ftp_password

ftp_url

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

Default value None

type str

triggers rerun False

Available in unodes

- get_ftp_files_1_0_0

Ursgal value translations for *ftp_url*

Style	Translation
get_http_style_1	ftp_url

header_translations

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

Default value None

type str

triggers rerun False

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
- msgfplus2csv_v2016_09_16
- msgfplus2csv_v2017_01_27
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- msfragger_20170103

Ursgal value translations for *header_translations*

Style	Translation
kojak_percolator_style_1	header_translations
msamanda_style_1	header_translations
msfragger_style_1	header_translations
msgfplus_style_1	header_translations
novor_style_1	header_translations
omssa_style_1	header_translations
pepnovo_style_1	header_translations

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_percolator_style_1	msamanda_style_1	msfra
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
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
Amanda Score	n/t	Amanda:Score	n/t
C-Gap	n/t	n/t	n/t
Charge	n/t	Charge	n/t
CumProb	n/t	n/t	n/t
DeNovoScore	n/t	n/t	n/t
Downstream Amino Acid	n/t	n/t	Seque
EValue	n/t	n/t	n/t
Filename	n/t	Filename	n/t
Hit rank	n/t	n/t	MSFr
Hyperscore	n/t	n/t	MSFr
Intercept of expectation model (expectation in log space)	n/t	n/t	MSFr
MSGFScore	n/t	n/t	n/t
Mass difference	n/t	n/t	MSFr
Matched fragment ions	n/t	n/t	MSFr
Modifications	n/t	Modifications	n/t
N-Gap	n/t	n/t	n/t
Neutral mass of peptide	n/t	n/t	MSFr
Next score	n/t	n/t	MSFr
Number of missed cleavages	n/t	n/t	MSFr
Number of tryptic termini	n/t	n/t	MSFr
PSMId	PSMId	n/t	n/t

Ursgal Value	Translated Value		
.	kojak_percolator_style_1	msamanda_style_1	msfra
Peptide	n/t	n/t	n/t
Peptide Sequence	n/t	n/t	Seque
PnvScr	n/t	n/t	n/t
Precursor	n/t	n/t	n/t
Precursor charge	n/t	n/t	Charg
Precursor neutral mass (Da)	n/t	n/t	MSFr
Protein	n/t	n/t	Protei
Protein Accessions	n/t	proteinacc_start_stop_pre_post;	n/t
RT	n/t	Retention Time (s)	n/t
Rank	n/t	Rank	n/t
Retention time (minutes)	n/t	n/t	Reten
RnkScr	n/t	n/t	n/t
Scan Number	n/t	Spectrum ID	n/t
ScanID	n/t	n/t	Spect
ScanNum	n/t	n/t	n/t
Sequence	n/t	Sequence	n/t
Slope of expectation model (expectation in log space)	n/t	n/t	MSFr
SpecEValue	n/t	n/t	n/t
SpecFile	n/t	n/t	n/t
Spectrum number	n/t	n/t	n/t
Title	n/t	Spectrum Title	n/t
Total possible number of matched theoretical fragment ions	n/t	n/t	MSFr
Upstream Amino Acid	n/t	n/t	Seque
Variable modifications detected	n/t	n/t	Modi
Weighted Probability	n/t	Amanda:Weighted Probability	n/t
[M+H]	n/t	n/t	n/t
m/z	n/t	Exp m/z	n/t
output_aa_probs	n/t	n/t	n/t
peptide	Sequence	n/t	n/t
posterior_error_prob	PEP	n/t	n/t
proteinIds	Protein ID	n/t	n/t
q-value	q-value	n/t	n/t
score	Kojak:score	n/t	n/t

heatmap_annotation_field_name

The name of the annotation to plot in the heatmap

Default value Protein

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_annotation_field_name*

Style	Translation
heatmap_style_1	heatmap_annotation_field_name

heatmap_box_style

Box style for the heatmap

Default value classic

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_box_style*

Style	Translation
heatmap_style_1	heatmap_box_style

heatmap_color_gradient

Color gradient for the heatmap

Default value Spectral

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_color_gradient*

Style	Translation
heatmap_style_1	heatmap_color_gradient

heatmap_column_order

The plot order of the columns

Default value []

type list

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_column_order*

Style	Translation
heatmap_style_1	heatmap_column_order

heatmap_error_suffix

The suffix to identify the value error holding columns

Default value _std

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_error_suffix*

Style	Translation
heatmap_style_1	heatmap_error_suffix

heatmap_identifier_field_name

The name of the identifier to plot in the heatmap

Default value Protein

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_identifier_field_name*

Style	Translation
heatmap_style_1	heatmap_identifier_field_name

heatmap_max_value

Maximum value for the color gradient

Default value 3

type int

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_max_value*

Style	Translation
heatmap_style_1	heatmap_max_value

heatmap_min_value

Minimum vaue for the color gradient

Default value -3

type int

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_min_value*

Style	Translation
heatmap_style_1	heatmap_min_value

heatmap_value_suffix

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

Default value _mean

type str

triggers rerun False

Available in unodes

- plot_pygcluster_heatmap_from_csv_1_0_0

Ursgal value translations for *heatmap_value_suffix*

Style	Translation
heatmap_style_1	heatmap_value_suffix

helper_extension

Exension for helper files

Default value .u.json

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *helper_extension*

Style	Translation
ucontroller_style_1	helper_extension

http_output_folder

Default http download path '' : None

Default value None

type str

triggers rerun False

Available in unodes

- get_http_files_1_0_0

Ursgal value translations for *http_output_folder*

Style	Translation
get_http_style_1	http_output_folder

http_url

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

Default value None

type str

triggers rerun False

Available in unodes

- get_http_files_1_0_0

Ursgal value translations for *http_url*

Style	Translation
get_http_style_1	http_url

instrument

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

Default value q_exactive

type select

triggers rerun False

Available in unodes

- kojak_1_5_3
- moda_v1_51
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- novor_1_1beta

Ursgal value translations for *instrument*

Style	Translation
kojak_style_1	instrument
moda_style_1	Instrument
msgfplus_style_1	-inst
novor_style_1	massAnalyzer

Ursgal key 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

intensity_cutoff

Low intensity cutoff as a fraction of max peak

Default value 0.0

type float

triggers rerun False

Available in unodes

- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *intensity_cutoff*

Style	Translation
msfragger_style_1	minimum_ratio
omssa_style_1	-cl

json_extension

Exension for .json files

Default value .u.json

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *json_extension*

Style	Translation
ucontroller_style_1	json_extension

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 False

Available in unodes

- unify_csv_1_0_0

Ursgal value translations for *keep_asp_pro_broken_peps*

Style	Translation
unify_csv_style_1	keep_asp_pro_broken_peps

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 False

Available in unodes

- svm_1_0_0

Ursgal value translations for *kernel*

Style	Translation
svm_style_1	kernel

kojak_MS1_centroid

MS1 centroided data yes (1) or no (0)

Default value No

type select

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_MS1_centroid*

Style	Translation
kojak_style_1	kojak_MS1_centroid

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
No	0
Yes	1

kojak_MS1_resolution

MS1 resolution

Default value 30000

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_MS1_resolution*

Style	Translation
kojak_style_1	kojak_MS1_resolution

kojak_MS2_centroid

MS2 centroided data yes (1) or no (0)

Default value Yes

type select

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_MS2_centroid*

Style	Translation
kojak_style_1	kojak_MS2_centroid

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
No	0
Yes	1

kojak_MS2_resolution

MS2 resolution

Default value 25000

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_MS2_resolution*

Style	Translation
kojak_style_1	kojak_MS2_resolution

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_diff_mods_on_xl*

Style	Translation
kojak_style_1	kojak_diff_mods_on_xl

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_enrichment*

Style	Translation
kojak_style_1	kojak_enrichment

kojak_export_pepxml

Activate (1) or deactivate (0) output as pepXML

Default value Deactivate

type select

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_export_pepxml*

Style	Translation
kojak_style_1	kojak_export_pepXML

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
Activate	1
Deactivate	0

kojak_export_percolator

Activate (1) or deactivate (0) output for percolator

Default value Activate

type select

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_export_percolator*

Style	Translation
kojak_style_1	kojak_export_percolator

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
Activate	1
Deactivate	0

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_fragment_bin_offset*

Style	Translation
kojak_style_1	kojak_fragment_bin_offset

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_fragment_bin_size*

Style	Translation
kojak_style_1	kojak_fragment_bin_size

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_mono_links_on_xl*

Style	Translation
kojak_style_1	kojak_mono_links_on_xl

kojak_percolator_version

Defines the output format of Kojak for Percolator

Default value 2.08

type str

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_percolator_version*

Style	Translation
kojak_style_1	kojak_percolator_version

kojak_prefer_precursor_pred

prefer precursor mono mass predicted by instrument software. 0 = ignore previous predictions 1 = use only previous predictions 2 = supplement predictions with additional analysis

Default value supplement

type select

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_prefer_precursor_pred*

Style	Translation
kojak_style_1	kojak_prefer_precursor_pred

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
0: ignore previous	0
1: only previous	1
2: supplement	2

kojak_spectrum_processing

True, if spectrum should be processed by kojak

Default value False

type bool

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_spectrum_processing*

Style	Translation
kojak_style_1	kojak_spectrum_processing

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

kojak_top_count

number of top scoring single peptides to combine in relaxed analysis

Default value 300

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_top_count*

Style	Translation
kojak_style_1	kojak_top_count

kojak_truncate_prot_names

Max protein name character to export, 0=off

Default value 0

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_truncate_prot_names*

Style	Translation
kojak_style_1	kojak_truncate_prot_names

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *kojak_turbo_button*

Style	Translation
kojak_style_1	kojak_turbo_button

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

label

15N if the corresponding amino acid labeling was applied

Default value 14N

type select

triggers rerun False

Available in unodes

- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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

Ursgal value translations for *label*

Style	Translation
moda_style_1	label
msamanda_style_1	label
msfragger_style_1	label
msgfplus_style_1	label
myrimatch_style_1	label
omssa_style_1	('-tem', '-tom')
xtandem_style_1	protein, modified residue mass file

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 False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *machine_offset_in_ppm*

Style	Translation
mzml2mgf_style_1	machine_offset_in_ppm

max_accounted_observed_peaks

Maximum number of peaks from a spectrum used.

Default value 100

type int

triggers rerun False

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

Ursgal value translations for *max_accounted_observed_peaks*

Style	Translation
kojak_style_1	max_accounted_observed_peaks
msfragger_style_1	use_topN_peaks
myrimatch_style_1	MaxPeakCount
xtandem_style_1	spectrum, total peaks

max_missed_cleavages

Maximum number of missed cleavages per peptide

Default value 2

type int

triggers rerun False

Available in unodes

- kojak_1_5_3
- 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
- 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

Ursgal value translations for *max_missed_cleavages*

Style	Translation
kojak_style_1	max_missed_cleavages
moda_style_1	MissedCleavage
msamanda_style_1	missed_cleavages
msfragger_style_1	allowed_missed_cleavage
myrimatch_style_1	MaxMissedCleavages
omssa_style_1	-v
unify_csv_style_1	max_missed_cleavages
upeptide_mapper_style_1	max_missed_cleavages
xtandem_style_1	scoring, maximum missed cleavage sites

max_mod_alternatives

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

Default value 6

type int

triggers rerun False

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *max_mod_alternatives*

Style	Translation
xtandem_style_1	protein, ptm complexity

max_mod_size

Minimum modification size to consider (in Da)

Default value 200

type int

triggers rerun False

Available in unodes

- moda_v1_51

Ursgal value translations for *max_mod_size*

Style	Translation
moda_style_1	MaxModSize

max_num_mods

Maximal number of modifications per peptide

Default value 3

type int

triggers rerun False

Available in unodes

- kojak_1_5_3
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *max_num_mods*

Style	Translation
kojak_style_1	max_num_mods
msgfplus_style_1	NumMods
myrimatch_style_1	MaxDynamicMods

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 False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *max_num_of_ions_per_series_to_search*

Style	Translation
omssa_style_1	-sp

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
all	0

max_num_per_mod

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

Default value 2

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *max_num_per_mod*

Style	Translation
msfragger_style_1	max_variable_mods_per_mod

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 False

Available in unodes

- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *max_num_per_mod_name_specific*

Style	Translation
xtandem_style_1	residue, potential modification mass

max_output_e_value

Highest e-value for reported peptides

Default value 1.0

type float

triggers rerun False

Available in unodes

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

Ursgal value translations for *max_output_e_value*

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

max_pep_length

Maximal length of a peptide

Default value 40

type int

triggers rerun False

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *max_pep_length*

Style	Translation
msfragger_style_1	digest_max_length
msgfplus_style_1	-maxLength
myrimatch_style_1	MaxPeptideLength
omssa_style_1	-nox

max_pep_var

Maximal peptide variants, new default defined by msfragger

Default value 1000

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- msfragger_20170103

Ursgal value translations for *max_pep_var*

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

mgf_input_file

Path to input .mgf file '' : None

Default value None

type str

triggers rerun False

Available in unodes

- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal value translations for *mgf_input_file*

Style	Translation
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
xtandem_style_1	spectrum, path

min_mod_size

Minimum modification size to consider (in Da)

Default value -200

type int

triggers rerun False

Available in unodes

- moda_v1_51

Ursgal value translations for *min_mod_size*

Style	Translation
moda_style_1	MinModSize

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 False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140
- pepnovo_3_1

Ursgal value translations for *min_output_score*

Style	Translation
myrimatch_style_1	MinResultScore
pepnovo_style_1	-min_filter_prob

Ursgal key translations

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

min_pep_length

Minimal length of a peptide

Default value 6

type int

triggers rerun False

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *min_pep_length*

Style	Translation
msfragger_style_1	digest_min_length
msgfplus_style_1	-minLength
myrimatch_style_1	MinPeptideLength
omssa_style_1	-no

min_precursor_matches

Minimum number of precursors that match a spectrum.

Default value 1

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *min_precursor_matches*

Style	Translation
omssa_style_1	-pc

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 False

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

Ursgal value translations for *min_required_matched_peaks*

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

min_required_observed_peaks

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

Default value 5

type int

triggers rerun False

Available in unodes

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

Ursgal value translations for *min_required_observed_peaks*

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

moda_blind_mode

Allowed number of modifications per peptide. '0' = no modification, '1' = one modification, '2' = no limit

Default value No Limit

type select

triggers rerun False

Available in unodes

- moda_v1_51

Ursgal value translations for *moda_blind_mode*

Style	Translation
moda_style_1	BlindMode

Ursgal key translations

Ursgal Value	Translated Value
.	moda_style_1
No Limit	2
No Modification	0
One Modification	1

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 False

Available in unodes

- moda_v1_51

Ursgal value translations for *moda_high_res*

Style	Translation
moda_style_1	HighResolution

Ursgal key translations

Ursgal Value	Translated Value
.	moda_style_1
0	OFF
1	ON

moda_protocol_id

MODa specific protocol to enable scoring parameters for labeled samples.

Default value None

type select

triggers rerun False

Available in unodes

- moda_v1_51

Ursgal value translations for *moda_protocol_id*

Style	Translation
moda_style_1	Protocol

Ursgal key translations

Ursgal Value	Translated Value
.	moda_style_1
None	NONE

modifications

Modifications are given as a list of strings, each representing the modification of one amino acid. The string consists of four info
'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 False

Available in unodes

- kojak_1_5_3
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- 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

Ursgal value translations for *modifications*

Style	Translation
kojak_style_1	modifications
moda_style_1	ADD
msamanda_style_1	modifications
msfragger_style_1	modifications
msgfplus_style_1	-mod
myrimatch_style_1	('DynamicMods', 'StaticMods')
novor_style_1	('variableModifications', 'fixedModifications')
omssa_style_1	('-mv', 'mf')
pepnovo_style_1	-PTMs
unify_csv_style_1	modifications
upeptide_mapper_style_1	modifications
xtandem_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')

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 False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *mono_link_definition*

Style	Translation
kojak_style_1	mono_link_definition

ms1_centroided

MS1 data are centroided: True or False

Default value False

type bool

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *ms1_centroided*

Style	Translation
kojak_style_1	kojak_MS1_centroid

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

ms1_resolution

MS1 resolution

Default value 30000

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *ms1_resolution*

Style	Translation
kojak_style_1	kojak_MS1_resolution

ms2_centroided

MS2 data are centroided: True or False

Default value True

type bool

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *ms2_centroided*

Style	Translation
kojak_style_1	kojak_MS2_centroid

Ursgal key translations

Ursgal Value	Translated Value
.	kojak_style_1
0	0
1	1

ms2_resolution

MS2 resolution

Default value 25000

type int

triggers rerun False

Available in unodes

- kojak_1_5_3

Ursgal value translations for *ms2_resolution*

Style	Translation
kojak_style_1	kojak_MS2_resolution

msfragger_add_topN_complementary

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

Default value 0

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_add_topN_complementary*

Style	Translation
msfragger_style_1	add_topN_complementary

msfragger_clear_mz_range

Removes peaks in this m/z range prior to matching. Useful for iTRAQ/TMT experiments (i.e. 0.0 150.0)

Default value 0.0 0.0

type str

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_clear_mz_range*

Style	Translation
msfragger_style_1	clear_mz_range

msfragger_min_fragments_modelling

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

Default value 3

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_min_fragments_modelling*

Style	Translation
msfragger_style_1	min_fragments_modelling

msfragger_output_max_expect

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

Default value 50

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_output_max_expect*

Style	Translation
msfragger_style_1	output_max_expect

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 False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_track_zero_topN*

Style	Translation
msfragger_style_1	track_zero_topN

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

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_zero_bin_accept_expect*

Style	Translation
msfragger_style_1	zero_bin_accept_expect

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

type int

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *msfragger_zero_bin_mult_expect*

Style	Translation
msfragger_style_1	zero_bin_mult_expect

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 False

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979

Ursgal value translations for *msgfplus_protocol_id*

Style	Translation
msgfplus_style_1	-protocol

Ursgal key translations

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

myrimatch_class_size_multiplier

Myrimatch ClassSizeMultiplier

Default value 2

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_class_size_multiplier*

Style	Translation
myrimatch_style_1	ClassSizeMultiplier

myrimatch_num_int_classes

Myrimatch NumIntensityClasses

Default value 3

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_num_int_classes*

Style	Translation
myrimatch_style_1	NumIntensityClasses

myrimatch_num_mz_fidelity_classes

Myrimatch NumMzFidelityClasses

Default value 3

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_num_mz_fidelity_classes*

Style	Translation
myrimatch_style_1	NumMzFidelityClasses

myrimatch_prot_sampl_time

Myrimatch ProteinSamplingTime

Default value 15

type int

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_prot_sampl_time*

Style	Translation
myrimatch_style_1	ProteinSamplingTime

myrimatch_smart_plus_three

Use Myrimatch UseSmartPlusThreeModel

Default value True

type bool

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_smart_plus_three*

Style	Translation
myrimatch_style_1	UseSmartPlusThreeModel

Ursgal key translations

Ursgal Value	Translated Value
.	myrimatch_style_1
0	0
1	1

myrimatch_tic_cutoff

Myrimatch TicCutoffPercentage

Default value 0.98

type float

triggers rerun False

Available in unodes

- myrimatch_2_1_138
- myrimatch_2_2_140

Ursgal value translations for *myrimatch_tic_cutoff*

Style	Translation
myrimatch_style_1	TicCutoffPercentage

mzidentml_compress

Compress mzidentml_lib output files

Default value False

type bool

triggers rerun False

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal value translations for *mzidentml_compress*

Style	Translation
mzidentml_style_1	-compress

Ursgal key translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

mzidentml_converter_version

mzidentml converter version: version name

Default value mzidentml_lib_1_6_10

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *mzidentml_converter_version*

Style	Translation
ucontroller_style_1	mzidentml_converter_version

mzidentml_export_type

Defines which paramters shoul be exporte by mzidentml_lib

Default value exportPSMs

type select

triggers rerun False

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal value translations for *mzidentml_export_type*

Style	Translation
mzidentml_style_1	-exportType

mzidentml_function

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

Default value Mzid2Csv

type select

triggers rerun False

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal value translations for *mzidentml_function*

Style	Translation
mzidentml_style_1	mzidentml_function

mzidentml_output_fragmentation

Include fragmentation in mzidentml_lib output

Default value False

type bool

triggers rerun False

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal value translations for *mzidentml_output_fragmentation*

Style	Translation
mzidentml_style_1	-outputFragmentation

Ursgal key translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

mzidentml_verbose_output

Verbose mzidentml_lib output

Default value False

type bool

triggers rerun False

Available in unodes

- mzidentml_lib_1_6_10
- mzidentml_lib_1_6_11
- mzidentml_lib_1_7

Ursgal value translations for *mzidentml_verbose_output*

Style	Translation
mzidentml_style_1	-verboseOutput

Ursgal key translations

Ursgal Value	Translated Value
.	mzidentml_style_1
0	false
1	true

mzml2mgf_converter_version

mzml to mgf converter version: version name

Default value mzml2mgf_1_0_0

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *mzml2mgf_converter_version*

Style	Translation
ucontroller_style_1	mzml2mgf_converter_version

neutral_loss_enabled

Neutral losses enabled for spectrum algorithm: set True or False

Default value False

type bool

triggers rerun False

Available in unodes

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

Ursgal value translations for *neutral_loss_enabled*

Style	Translation
xtandem_style_1	spectrum, use neutral loss window

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

neutral_loss_mass

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

Default value 0

type int

triggers rerun False

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance

- xtandem_alanine

Ursgal value translations for *neutral_loss_mass*

Style	Translation
xtandem_style_1	spectrum, neutral loss mass

neutral_loss_window

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

Default value 0

type int

triggers rerun False

Available in unodes

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

Ursgal value translations for *neutral_loss_window*

Style	Translation
xtandem_style_1	spectrum, neutral loss window

noise_suppression_enabled

Used noise suppresssion

Default value False

type bool

triggers rerun False

Available in unodes

- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer

Ursgal value translations for *noise_suppression_enabled*

Style	Translation
xtandem_style_1	spectrum, use noise suppression

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

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 False

Available in unodes

- sanitize_csv_1_0_0

Ursgal value translations for *num_compared_psms*

Style	Translation
sanitize_csv_style_1	num_compared_psms

num_hits_retain_spec

Maximum number of hits retained per precursor charge state per spectrum during the search

Default value 30

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *num_hits_retain_spec*

Style	Translation
omssa_style_1	-hl

num_i_decimals

Number of decimals for intensity (peak)

Default value 5

type int

triggers rerun False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *num_i_decimals*

Style	Translation
mzml2mgf_style_1	number_of_i_decimals

num_match_spec

Maximum number of peptide spectrum matches to report for each spectrum

Default value 10

type int

triggers rerun False

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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- pepnovo_3_1
- msfragger_20170103

Ursgal value translations for *num_match_spec*

Style	Translation
msamanda_style_1	max_rank
msfragger_style_1	output_report_topN
msgfplus_style_1	-n
myrimatch_style_1	MaxResultRank
omssa_style_1	-hc
pepnovo_style_1	-num_solutions

num_mz_decimals

Number of decimals for m/z mass

Default value 5

type int

triggers rerun False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *num_mz_decimals*

Style	Translation
mzml2mgf_style_1	number_of_mz_decimals

omssa_cp

Omssa: eliminate charge reduced precursors in spectra

Default value False

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_cp*

Style	Translation
omssa_style_1	-cp

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	0
1	1

omssa_h1

Omssa: number of peaks allowed in single charge window

Default value 2

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_h1*

Style	Translation
omssa_style_1	-h1

omssa_h2

Omssa: number of peaks allowed in double charge window

Default value 2

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_h2*

Style	Translation
omssa_style_1	-h2

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 False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_ht*

Style	Translation
omssa_style_1	-ht

omssa_mm

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

Default value 128

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_mm*

Style	Translation
omssa_style_1	-mm

omssa_ta

Omssa: automatic mass tolerance adjustment fraction

Default value 1.0

type float

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_ta*

Style	Translation
omssa_style_1	-ta

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 False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_tex*

Style	Translation
omssa_style_1	-tex

omssa_verbose

Omssa: verbose info print

Default value False

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_verbose*

Style	Translation
omssa_style_1	-ni

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	
1	-ni

omssa_w1

Omssa: single charge window in Da

Default value 27

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_w1*

Style	Translation
omssa_style_1	-w1

omssa_w2

Omssa: double charge window in Da

Default value 14

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_w2*

Style	Translation
omssa_style_1	-w2

omssa_z1

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

Default value 0.95

type float

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_z1*

Style	Translation
omssa_style_1	-z1

omssa_zc

Should charge plus one be determined algorithmically?

Default value True

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_zc*

Style	Translation
omssa_style_1	-zc

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	0
1	1

omssa_zcc

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

Default value 2

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_zcc*

Style	Translation
omssa_style_1	-zcc

omssa_zt

Minimum precursor charge to start considering multiply charged products

Default value 3

type int

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *omssa_zt*

Style	Translation
omssa_style_1	-zt

output_aa_probs

Output probabilities for each amino acid.

Default value True

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *output_aa_probs*

Style	Translation
pepnovo_style_1	-output_aa_probs

output_add_features

Number of decimals for intensity (peak)

Default value True

type bool

triggers rerun False

Available in unodes

- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979

Ursgal value translations for *output_add_features*

Style	Translation
msgfplus_style_1	-addFeatures

Ursgal key translations

Ursgal Value	Translated Value
.	msgfplus_style_1
0	0
1	1

output_cum_probs

Output cumulative probabilities.

Default value True

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *output_cum_probs*

Style	Translation
pepnovo_style_1	-output_cum_probs

output_file_incl_path

Path to output file 'None' : None

Default value None

type str

triggers rerun False

Available in unodes

- generate_target_decoy_1_0_0
- merge_csvs_1_0_0
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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
- omssa_2_1_9
- pepnovo_3_1
- percolator_2_08
- qvality_2_02
- venndiagram_1_0_0
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine

Ursgal value 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
qvality_style_1	-o
venndiagram_style_1	output_file
xtandem_style_1	output, path

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 False

Available in unodes

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

Ursgal value translations for *output_file_type*

Style	Translation
omssa_style_1	('-oc', '-ox')
xtandem_style_1	output, mzid

Ursgal key translations

Ursgal Value	Translated Value	
.	omssa_style_1	xtandem_style_1
.csv	-oc	n/t
.mzid	n/t	yes
.omx	-ox	n/t
default	-oc	no

output_prm

Only print spectrum graph nodes with scores.

Default value False

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *output_prm*

Style	Translation
pepnovo_style_1	-prm

output_prm_norm

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

Default value False

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *output_prm_norm*

Style	Translation
pepnovo_style_1	-prm_norm

output_q_values

Output Q-values

Default value True

type bool

triggers rerun False

Available in unodes

- msgfplus2csv_v2016_09_16
- msgfplus2csv_v2017_01_27

Ursgal value translations for *output_q_values*

Style	Translation
msgfplus_style_1	-showQValue

Ursgal key translations

Ursgal Value	Translated Value
.	msgfplus_style_1
0	0
1	1

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 False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *pepnovo_tag_length*

Style	Translation
pepnovo_style_1	-tag_length

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_v3

type str

triggers rerun False

Available in unodes

- upeptide_mapper_1_0_0

Ursgal value translations for *peptide_mapper_class_version*

Style	Translation
upeptide_mapper_style_1	peptide_mapper_class_version

peptide_mapper_converter_version

determines which upeptide mapper node should be used

Default value upeptide_mapper_1_0_0

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *peptide_mapper_converter_version*

Style	Translation
ucontroller_style_1	peptide_mapper_converter_version

precursor_charge_dependency

charge dependency of precursor mass tolerance (none or linear)

Default value linear

type select

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *precursor_charge_dependency*

Style	Translation
omssa_style_1	-tez

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
linear	1
none	0

precursor_isotope_range

Error range for incorrect carbon isotope parent ion assignment

Default value 0,1

type select

triggers rerun False

Available in unodes

- kojak_1_5_3
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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

Ursgal value translations for *precursor_isotope_range*

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

Ursgal key translations

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

precursor_mass_tolerance_minus

Precursor mass tolerance: lower mass tolerance of measured and calculated parent ion M+H

Default value 5

type int

triggers rerun False

Available in unodes

- kojak_1_5_3
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138

- myrimatch_2_2_140
- novor_1_1beta
- 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

Ursgal value translations for *precursor_mass_tolerance_minus*

Style	Translation
kojak_style_1	ppm_tolerance_pre
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol
msfragger_style_1	precursor_mass_tolerance
msgfplus_style_1	-t
myrimatch_style_1	MonoPrecursorMzTolerance
novor_style_1	precursorErrorTol
omssa_style_1	-te
pepnovo_style_1	-pm_tolerance
unify_csv_style_1	precursor_mass_tolerance_minus
xtandem_style_1	spectrum, parent monoisotopic mass error minus

precursor_mass_tolerance_plus

Precursor mass tolerance: higher mass tolerance of measured and calculated parent ion M+H

Default value 5

type int

triggers rerun False

Available in unodes

- kojak_1_5_3
- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- 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

Ursgal value translations for *precursor_mass_tolerance_plus*

Style	Translation
kojak_style_1	ppm_tolerance_pre
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol
msfragger_style_1	precursor_mass_tolerance
msgfplus_style_1	-t
myrimatch_style_1	MonoPrecursorMzTolerance
novor_style_1	precursorErrorTol
omssa_style_1	-te
pepnovo_style_1	-pm_tolerance
unify_csv_style_1	precursor_mass_tolerance_minus
xtandem_style_1	spectrum, parent monoisotopic mass error plus

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 False

Available in unodes

- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- novor_1_1beta
- omssa_2_1_9
- pepnovo_3_1
- xtandem_cyclone_2010
- xtandem_jackhammer
- xtandem_piledriver
- xtandem_sledgehammer
- xtandem_vengeance
- xtandem_alanine
- msfragger_20170103

Ursgal value translations for *precursor_mass_tolerance_unit*

Style	Translation
moda_style_1	PPMTolerance
msamanda_style_1	ms1_tol unit
msfragger_style_1	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
xtandem_style_1	spectrum, parent monoisotopic mass error units

Ursgal key translations

Ursgal Value	Translated Value						
.	msamanda_style_1	myri-match_style_1	msgf-plus_style_1	myri-match_style_1	novor_style_1	omssa_style_1	x!tan-dem_style_1
da	Da	0	Da	Da	Da		Daltons
ppm	n/t	1	n/t	n/t	n/t	-teppm	n/t

precursor_mass_type

Precursor mass type: monoisotopic or average

Default value monoisotopic

type select

triggers rerun False

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
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9

Ursgal value translations for *precursor_mass_type*

Style	Translation
msamanda_style_1	monoisotopic
myrimatch_style_1	PrecursorMzToleranceRule
omssa_style_1	-tem

Ursgal key translations

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

precursor_max_charge

Maximal accepted parent ion charge

Default value 5

type int

triggers rerun False

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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- myrimatch_2_1_138
- myrimatch_2_2_140
- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *precursor_max_charge*

Style	Translation
msamanda_style_1	considered_charges
msfragger_style_1	precursor_max_charge
msgfplus_style_1	-maxCharge
myrimatch_style_1	NumChargeStates
omssa_style_1	-zh

precursor_max_mass

Maximal parent ion mass. Adjusted to default used by MSFragger

Default value 7000

type int

triggers rerun False

Available in unodes

- kojak_1_5_3
- myrimatch_2_1_138
- myrimatch_2_2_140
- msfragger_20170103

Ursgal value translations for *precursor_max_mass*

Style	Translation
kojak_style_1	precursor_max_mass
msfragger_style_1	precursor_max_mass
myrimatch_style_1	MaxPeptideMass
xtandem_style_1	spectrum, minimum parent m+h

precursor_min_charge

Minimal accepted parent ion charge

Default value 1

type int

triggers rerun False

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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- msgfplus_v9979
- omssa_2_1_9
- msfragger_20170103

Ursgal value translations for *precursor_min_charge*

Style	Translation
msamanda_style_1	considered_charges
msfragger_style_1	precursor_min_charge
msgfplus_style_1	-minCharge
omssa_style_1	-zl

precursor_min_mass

Minimal parent ion mass

Default value 400

type int

triggers rerun False

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

Ursgal value translations for *precursor_min_mass*

Style	Translation
kojak_style_1	precursor_min_mass
msfragger_style_1	precursor_min_mass
myrimatch_style_1	MinPeptideMass
xtandem_style_1	spectrum, minimum parent m+h

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 False

Available in unodes

- msfragger_20170103

Ursgal value translations for *precursor_true_tolerance*

Style	Translation
msfragger_style_1	precursor_true_tolerance

precursor_true_units

Mass tolerance units fo precursor_true_tolerance

Default value ppm

type str

triggers rerun False

Available in unodes

- msfragger_20170103

Ursgal value translations for *precursor_true_units*

Style	Translation
msfragger_style_1	precursor_true_units

Ursgal key translations

Ursgal Value	Translated Value
.	msfragger_style_1
da	0
ppm	1

prefix

'None' : None

Default value None

type None

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *prefix*

Style	Translation
ucontroller_style_1	prefix

protein_delimiter

This delimiter seperates protein IDs/names in the unified csv

Default value <|>

type str

triggers rerun False

Available in unodes

- percolator_2_08
- unify_csv_1_0_0
- upeptide_mapper_1_0_0

Ursgal value translations for *protein_delimiter*

Style	Translation
percolator_style_1	protein_delimiter
unify_csv_style_1	protein_delimiter
upeptide_mapper_style_1	protein_delimiter

psm_merge_delimiter

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

Default value ;

type str

triggers rerun False

Available in unodes

- unify_csv_1_0_0

Ursgal value translations for *psm_merge_delimiter*

Style	Translation
unify_csv_style_1	psm_merge_delimiter

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 False

Available in unodes

- qvality_2_02

Ursgal value translations for *qvality_cross_validation*

Style	Translation
qvality_style_1	-c

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 False

Available in unodes

- qvality_2_02

Ursgal value translations for *qvality_epsilon_step*

Style	Translation
qvality_style_1	-s

qvality_number_of_bins

Number of bins used in qvality

Default value 500

type int

triggers rerun False

Available in unodes

- qvality_2_02

Ursgal value translations for *qvality_number_of_bins*

Style	Translation
qvality_style_1	-n

qvality_verbose

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

Default value 2

type select

triggers rerun False

Available in unodes

- qvality_2_02

Ursgal value translations for *qvality_verbose*

Style	Translation
qvality_style_1	-v

Ursgal key translations

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

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 False

Available in unodes

- ucontroller

Ursgal value translations for *raw_ident_csv_suffix*

Style	Translation
ucontroller_style_1	raw_ident_csv_suffix

remove_redundant_psms

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'.

Default value True

type bool

triggers rerun False

Available in unodes

- sanitize_csv_1_0_0

Ursgal value translations for *remove_redundant_psms*

Style	Translation
sanitize_csv_style_1	remove_redundant_psms

remove_temporary_files

Remove temporary files: True or False

Default value True

type bool

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *remove_temporary_files*

Style	Translation
ucontroller_style_1	remove_temporary_files

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 False

Available in unodes

- unify_csv_1_0_0

Ursgal value translations for *rounded_mass_decimals*

Style	Translation
unify_csv_style_1	rounded_mass_decimals

rt_pickle_name

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

Default value _ursgal_lookup.pkl

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *rt_pickle_name*

Style	Translation
ucontroller_style_1	rt_pickle_name

sanitize_csv_converter_version

sanitize csv converter version: version name

Default value sanitize_csv_1_0_0

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *sanitize_csv_converter_version*

Style	Translation
ucontroller_style_1	sanitize_csv_converter_version

scan_exclusion_list

Spectra rejected during mzml2mgf conversion

Default value []

type list

triggers rerun False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *scan_exclusion_list*

Style	Translation
mzml2mgf_style_1	scan_exclusion_list

scan_inclusion_list

Exclusively spectra included during mzml2mgf conversion

Default value None

type list

triggers rerun False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *scan_inclusion_list*

Style	Translation
mzml2mgf_style_1	scan_inclusion_list

scan_skip_modulo_step

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

Default value None

type int

triggers rerun False

Available in unodes

- mzml2mgf_1_0_0

Ursgal value translations for *scan_skip_modulo_step*

Style	Translation
mzml2mgf_style_1	scan_skip_modulo_step

Ursgal key translations

Ursgal Value	Translated Value
.	mzml2mgf_style_1

score-h2o_ions

Spectrum: if true, ions loss of H2O are respected in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_-h2o_ions*

Style	Translation
msamanda_style_1	series

score-nh3_ions

Spectrum: if true, ions loss of NH3 are respected in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_-nh3_ions*

Style	Translation
msamanda_style_1	series

score_a_ions

Spectrum: if true, a ions are used in algorithm

Default value False

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_a_ions*

Style	Translation
kojak_style_1	ion_series_A
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, a ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	0	yes

score_b1_ions

first forward (b1) product ions inclued in search

Default value False

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *score_b1_ions*

Style	Translation
omssa_style_1	-sb1

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	1
1	0

score_b_ions

Spectrum: if true, b ions are used in algorithm

Default value True

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_b_ions*

Style	Translation
kojak_style_1	ion_series_B
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, b ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	1	yes

score_c_ions

Spectrum: if true, c ions are used in algorithm

Default value False

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_c_ions*

Style	Translation
kojak_style_1	ion_series_C
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, c ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	2	yes

score_c_terminal_ions

Score c terminal ions

Default value True

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *score_c_terminal_ions*

Style	Translation
omssa_style_1	-sct

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	1
1	0

score_correlation_corr

Use correlation correction to score?

Default value True

type bool

triggers rerun False

Available in unodes

- omssa_2_1_9

Ursgal value translations for *score_correlation_corr*

Style	Translation
omssa_style_1	-scorr

Ursgal key translations

Ursgal Value	Translated Value
.	omssa_style_1
0	1
1	0

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 False

Available in unodes

- sanitize_csv_1_0_0

Ursgal value translations for *score_diff_threshold*

Style	Translation
sanitize_csv_style_1	score_diff_threshold

score_imm_ions

Spectrum: if true, immonium ions are respected in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_imm_ions*

Style	Translation
msamanda_style_1	series

score_int_ions

Spectrum: if true, internal fragment ions are respect in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_int_ions*

Style	Translation
msamanda_style_1	series

score_x_ions

Spectrum: if true, x ions are used in algorithm

Default value False

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_x_ions*

Style	Translation
kojak_style_1	ion_series_X
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, x ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	3	yes

score_y_ions

Spectrum: if true, y ions are used in algorithm

Default value True

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_y_ions*

Style	Translation
kojak_style_1	ion_series_Y
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, y ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	4	yes

score_z+1_ions

Spectrum: if true, z ion plus 1 Da mass are used in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_z+1_ions*

Style	Translation
msamanda_style_1	series

score_z+2_ions

Spectrum: if true z ion plus 2 Da mass are used in algorithm

Default value False

type bool

triggers rerun False

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

Ursgal value translations for *score_z+2_ions*

Style	Translation
msamanda_style_1	series

score_z_ions

Spectrum: if true, z ions are used in algorithm

Default value False

type bool

triggers rerun False

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
- 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

Ursgal value translations for *score_z_ions*

Style	Translation
kojak_style_1	ion_series_Z
msamanda_style_1	series
myrimatch_style_1	FragmentationRule
omssa_style_1	-i
xtandem_style_1	scoring, z ions

Ursgal key translations

Ursgal Value	Translated Value		
.	kojak_style_1	omssa_style_1	xtandem_style_1
0	0		no
1	1	5	yes

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 False

Available in unodes

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

Ursgal value translations for *search_for_saps*

Style	Translation
xtandem_style_1	protein, saps

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

semi_enzyme

Allows semi-enzymatic peptide ends

Default value False

type bool

triggers rerun False

Available in unodes

- 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
- msgfplus_v2016_09_16
- msgfplus_v2017_01_27
- 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

Ursgal value 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
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 key translations

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

show_unodes_in_development

Show ursgal nodes that are in development: False or True

Default value False

type bool

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *show_unodes_in_development*

Style	Translation
ucontroller_style_1	show_unodes_in_development

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 False

Available in unodes

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

Ursgal value translations for *spec_dynamic_range*

Style	Translation
xtandem_style_1	spectrum, dynamic range

svm_c_param

Penalty parameter C of the error term of the post-processing SVM

Default value 1.0

type float

triggers rerun False

Available in unodes

- svm_1_0_0

Ursgal value translations for *svm_c_param*

Style	Translation
svm_style_1	c

test_param1

TEST/DEBUG: Internal Ursgal parameter 1 for debugging and testing.

Default value b

type select

triggers rerun False

Available in unodes

- _test_node

Ursgal value translations for *test_param1*

Style	Translation
_test_node_style_1	test_param1

Ursgal key translations

Ursgal Value	Translated Value
.	_test_node_style_1
a	A
b	B
c	C
d	D
e	E

test_param2

TEST/DEBUG: Internal Ursgal parameter 2 for debugging and testing.

Default value three

type select

triggers rerun False

Available in unodes

- _test_node

Ursgal value translations for *test_param2*

Style	Translation
_test_node_style_1	test_param2

Ursgal key translations

Ursgal Value	Translated Value
.	_test_node_style_1
five	5
four	4
one	1
three	3
two	2

threshold_is_log10

True, if log10 scale has been used for score_diff_threshold.

Default value True

type bool

triggers rerun False

Available in unodes

- sanitize_csv_1_0_0

Ursgal value translations for *threshold_is_log10*

Style	Translation
sanitize_csv_style_1	threshold_is_log10

unify_csv_converter_version

unify csv converter version: version name

Default value unify_csv_1_0_0

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *unify_csv_converter_version*

Style	Translation
ucontroller_style_1	unify_csv_converter_version

ursgal_resource_url

URL that is used to install and prepare_resources.py

Default value http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/

type str

triggers rerun False

Available in unodes

- ucontroller

Ursgal value translations for *ursgal_resource_url*

Style	Translation
ucontroller_style_1	ursgal_resource_url

use_quality_filter

Use filter for low quality spectra.

Default value True

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1

Ursgal value translations for *use_quality_filter*

Style	Translation
pepnovo_style_1	-no_quality_filter

Ursgal key translations

Ursgal Value	Translated Value
.	pepnovo_style_1
0	1
1	0

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 False

Available in unodes

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

Ursgal value translations for *use_refinement*

Style	Translation
xtandem_style_1	refine

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

use_spectrum_charge

Does not correct precursor charge.

Default value True

type bool

triggers rerun False

Available in unodes

- pepnovo_3_1
- msfragger_20170103

Ursgal value translations for *use_spectrum_charge*

Style	Translation
msfragger_style_1	override_charge
pepnovo_style_1	-use_spectrum_charge

Ursgal key translations

Ursgal Value	Translated Value
.	msfragger_style_1
0	1
1	0

use_spectrum_mz

Does not correct precusor m/z.

Default value True

type bool

triggers rerun False

Available in unodes

- moda_v1_51
- pepnovo_3_1

Ursgal value translations for *use_spectrum_mz*

Style	Translation
moda_style_1	AutoPMCorrection
pepnovo_style_1	-use_spectrum_mz

Ursgal key translations

Ursgal Value	Translated Value
.	moda_style_1
0	1
1	0

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 False

Available in unodes

- ucontroller

Ursgal value translations for *validated_ident_csv_suffix*

Style	Translation
ucontroller_style_1	validated_ident_csv_suffix

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 False

Available in unodes

- qvality_2_02

Ursgal value translations for *validation_generalized*

Style	Translation
qvality_style_1	-g

Ursgal key translations

Ursgal Value	Translated Value
.	qvality_style_1
0	None
1	

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 False

Available in unodes

- qvality_2_02

Ursgal value translations for *validation_minimum_score*

Style	Translation
qvality_style_1	validation_minimum_score

Ursgal key translations

Ursgal Value	Translated Value
.	qvality_style_1
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
msfragger_20170103	0
msgfplus_v2016_09_16	1e-100
msgfplus_v2017_01_27	1e-100
msgfplus_v9979	1e-100
myrimatch_2_1_138	0
myrimatch_2_2_140	0
omssa_2_1_9	1e-30
xtandem_alanine	0
xtandem_cyclone_2010	0
xtandem_jackhammer	0
xtandem_piledriver	0
xtandem_sledgehammer	0
xtandem_vengeance	0

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 False

Available in unodes

- add_estimated_fdr_1_0_0
- percolator_2_08
- qvality_2_02
- sanitize_csv_1_0_0
- svm_1_0_0
- ucontroller
- unify_csv_1_0_0

- msfragger_20170103

Ursgal value translations for *validation_score_field*

Style	Translation
add_estimated_fdr_style_1	validation_score_field
msfragger_style_1	validation_score_field
percolator_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 key translations

Ursgal Value	Translated Value						
.	add_estimated_fdr_style_1	default_style_1	qvalitor_style_1	validity_style_1	sanitize_csv_style_1	svm_style_1	ucontroller_style_1	unify_csv_style_1
msamanda_1_0_0_13242_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_13243_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_14299_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_14300_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_15583_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msamanda_1_0_0_15584_core	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score	Amanda:Score
msfragger_20170103	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore	MSFragger:Hyperscore
msgfplus_v2016_09_16	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue
msgfplus_v2017_07_27	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue
msgfplus_v9979	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue	MS-GF:SpecEValue
myrimatch_2_1_138	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH
myrimatch_2_2_140	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH	MyriMatch:MVH
novor_1_1beta	Novor:score	Novor:score	Novor:score	Novor:score	Novor:score	Novor:score	Novor:score
omssa_2_1_9	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue	OMSSA:pvalue
pepnovo_3_1	PepNovo:PnvScr	PepNovo:PnvScr	PepNovo:PnvScr	PepNovo:PnvScr	PepNovo:PnvScr	PepNovo:PnvScr	PepNovo:PnvScr
xtandem_alanine	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore
xtandem_cyclone_2010	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore
xtandem_jackhammer	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore
xtandem_piledriver	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore
xtandem_sledgehammer	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore
xtandem_vengeance	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore	X!Tandem:hyperscore

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 False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_column_names*

Style	Translation
venndiagram_style_1	visualization_column_names

visualization_font

Font used for visualization plots (e.g. Venn diagram), given as tuple (font-type, font-size header, font-size major, font-size minor, font-size venn)

Default value ('Helvetica', 31, 25, 20, 20)

type tuple

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_font*

Style	Translation
venndiagram_style_1	visualization_font

visualization_header

Header of visualization output (e.g. Venn diagram)

Default value ursgal Venn Diagram

type str

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_header*

Style	Translation
venndiagram_style_1	header

visualization_label_list

Specifies labels for the datasets that should be visualized. Needs to be given in the same order as the datasets.

Default value []

type list

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_label_list*

Style	Translation
venndiagram_style_1	visualization_label_list

visualization_opacity

Opacity used in visualization plots (e.g. Venn diagram)

Default value 0.35

type float

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_opacity*

Style	Translation
venndiagram_style_1	opacity

visualization_scaling_factors

Scaling factor for visualization plots (e.g. Venn diagram), given as tuple (x-axis-scaling-factor, y-axis-scaling-factor)

Default value (600, 400)

type tuple

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_scaling_factors*

Style	Translation
venndiagram_style_1	visualization_scaling_factors

visualization_size

Size of visualization plots (e.g. Venn diagram), given as tuple (width, height)

Default value (1200, 900)

type tuple

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_size*

Style	Translation
venndiagram_style_1	visualization_size

visualization_stroke_width

Stroke width used in visualization plots (e.g. Venn diagram)

Default value 2.0

type float

triggers rerun False

Available in unodes

- venndiagram_1_0_0

Ursgal value translations for *visualization_stroke_width*

Style	Translation
venndiagram_style_1	stroke-width

window_size

Combined PEPs are computed by iterating a sliding window over the sorted PSMs. Each PSM receives a PEP based on the target/decoy ratio of the surrounding PEPs. This parameter defines the window size.

Default value 249

type int

triggers rerun False

Available in unodes

- combine_pep_1_0_0

Ursgal value translations for *window_size*

Style	Translation
combine_pep_style_1	window_size

word_len

word length used to index peptide mapper, smaller word len requires more memory

Default value 6

type int

triggers rerun False

Available in unodes

- unify_csv_1_0_0
- upeptide_mapper_1_0_0

Ursgal value translations for *word_len*

Style	Translation
unify_csv_style_1	word_len
upeptide_mapper_style_1	word_len

write_unfiltered_results

Writes rejected results if True

Default value False

type bool

triggers rerun False

Available in unodes

- filter_csv_1_0_0

Ursgal value translations for *write_unfiltered_results*

Style	Translation
filter_csv_style_1	write_unfiltered_results

xtandem_stp_bias

Interpretation of peptide phosphorylation models.

Default value False

type bool

triggers rerun False

Available in unodes

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

Ursgal value translations for *xtandem_stp_bias*

Style	Translation
xtandem_style_1	protein, stP bias

Ursgal key translations

Ursgal Value	Translated Value
.	xtandem_style_1
0	no
1	yes

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.

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 python3.4
# encoding: utf-8

import ursgal
import os
import sys
import shutil


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
```

```
            )

    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',
    )
    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 python3.4
# 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',
    ]

    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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/
↪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

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

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 python3.4
# 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  = [
        '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,
                )
                validated_results.append( validated_csv )

            validated_results_from_all_engines = uc.merge_csvs(
                    input_files = validated_results,
                )
            filtered_validated_results = uc.filter_csv(
                input_file = validated_results_from_all_engines,
                )
            result_files.append(filtered_validated_results)

        results_all_files = uc.merge_csvs(
            input_files = result_files,
        )

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print(main.__doc__)
        exit()
    main(
        folder                = sys.argv[1],
        profile               = sys.argv[2],
        target_decoy_database = sys.argv[3],
    )
```

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 5 XTandem versions are incorporated in Ursgal.

```
#!/usr/bin/env python3.4
# 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 5 XTandem versions are incorporated in Ursgal.

    '''
    engine_list = [
        'xtandem_cyclone',
        'xtandem_jackhammer',
        'xtandem_sledgehammer',
        'xtandem_piledriver',
        'xtandem_vengeance'

    ]

    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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/
↪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'
        )

    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.filter_csv(
            input_file = validated_file,
        )

        filtered_files_list.append( filtered_file )

    uc.visualize(
        input_files     = filtered_files_list,
        engine          = 'venndiagram',
    )
    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)

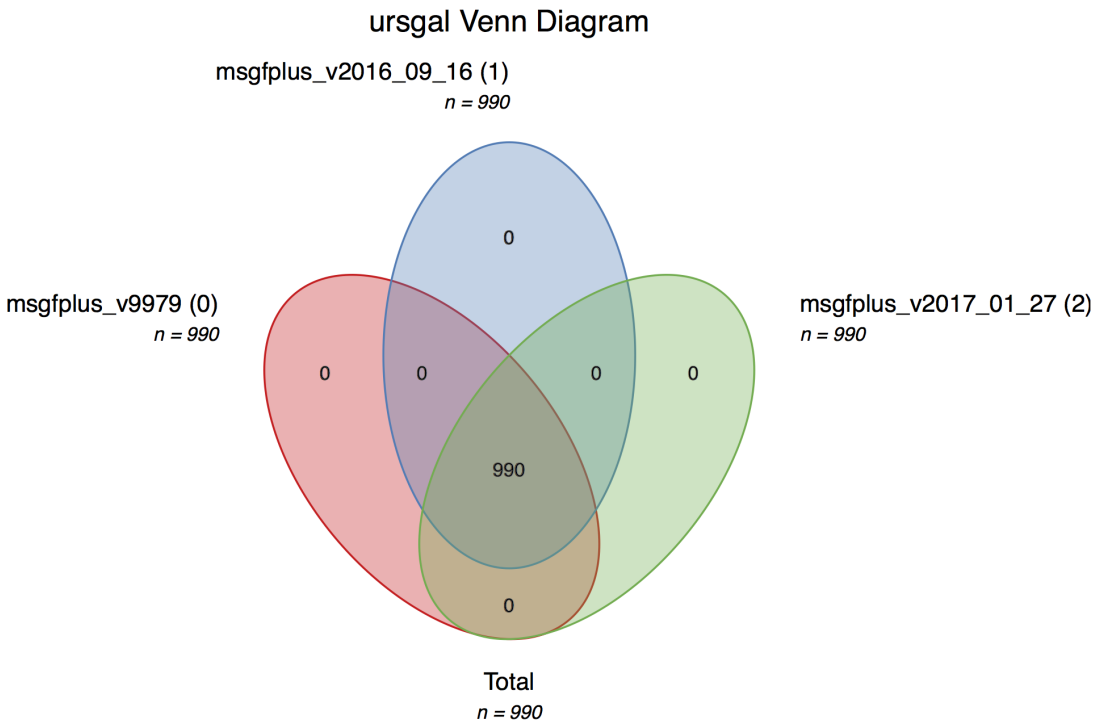
usage: ./msgf_version_comparison.py

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



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 python3.4
# 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 = [
        (-10e-6 , '-10_ppm_offset') ,
        (-9e-6  , '-9_ppm_offset')  ,
        (-8e-6  , '-8_ppm_offset')  ,
        (-7e-6  , '-7_ppm_offset')  ,
        (-6e-6  , '-6_ppm_offset')  ,
        (-5e-6  , '-5_ppm_offset')  ,
        (-4e-6  , '-4_ppm_offset')  ,
        (-3e-6  , '-3_ppm_offset')  ,
        (-2e-6  , '-2_ppm_offset')  ,
        (-1e-6  , '-1_ppm_offset')  ,
        (None   , '0_ppm_offset')   ,
        (1e-6   , '1_ppm_offset')   ,
        (2e-6   , '2_ppm_offset')   ,
        (3e-6   , '3_ppm_offset')   ,
        (4e-6   , '4_ppm_offset')   ,
        (5e-6   , '5_ppm_offset')   ,
        (6e-6   , '6_ppm_offset')   ,
        (7e-6   , '7_ppm_offset')   ,
        (8e-6   , '8_ppm_offset')   ,
        (9e-6   , '9_ppm_offset')   ,
        (10e-6  , '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,
            )

    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()
```

Bruderer et al. (2015) high resolution machine ppm offset example

qexactive_xtandem_version_comparison.**main**(*folder*)

Executes a search with 4 versions of X!Tandem on an example file from the data from Bruderer et al.

usage: ./xtandem_version_comparison.py

This is a simple example file to show the straightforward comparison of different program versions of X!Tandem when analyzing hish resolution data whihc 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 python3.4
# encoding: utf-8

import ursgal
import os
import sys


def main(folder):
    '''
    Executes a search with 4 versions of X!Tandem on an example file from the
    data from Bruderer et al.

    usage:
        ./xtandem_version_comparison.py


    This is a simple example file to show the straightforward comparison of
    different program versions of X!Tandem when analyzing hish resolution data
    whihc 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',
            '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'
        )
        exit()


    engine_list = [
        'xtandem_cyclone',
        'xtandem_jackhammer',
        'xtandem_sledgehammer',
        'xtandem_piledriver',
    ]

    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.filter_csv(
            input_file = validated_file,
        )

        filtered_files_list.append( filtered_file )

    uc.visualize(
        input_files     = filtered_files_list,
        engine          = 'venndiagram',
    )
    return

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print(main.__doc__)
        exit()
    main(sys.argv[1])
```

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 python3.4
# encoding: utf-8

import ursgal
import os


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.generate_target_decoy(
        input_files       = fasta_database_list,
        output_file_name = 'my_BSA_target_decoy.fasta',
    )
    print('Generated target decoy database: {0}'.format(new_target_decoy_db_name))

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 python3.4
# 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 ,
        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_to_mgf_and_update_rt_lookup(
        input_file = mzML_file
    )

    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,
            )

    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 python3.4
# 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,
        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_to_mgf_and_update_rt_lookup(
        input_file = mzML_file
    )

    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,
                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()
```

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 python3.4
# 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 = {
        '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.filter_csv(
        input_file = csv_file_to_filter,
    )


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 python3.4
# 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(
        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.filter_csv(
        input_file = csv_file_to_filter,

    )


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

Machine ppm offset sweep

Reproduce 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 python3.4
# 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):
    '''
    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 / 1e6 )

    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
            R.params['prefix']                = 'ppm_offset_{0}'.format(
                int(ppm_offset * 1e6)
            )
            mgf_file = R.convert_to_mgf_and_update_rt_lookup(
                input_file = mzML_path
            )
```

```
            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()
        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'         : '',
                        '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__)
        exit()
    search(sys.argv[1])
    analyze(sys.argv[1])
```

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 python3.4
# 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
    '''

    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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/
↪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',
            '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',
        ],
        ('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 )

            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.merge_csvs( unified_results_list )

            # 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'] = [
        ['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.merge_csvs( percolator_validated_list )

        twelve_filtered = []
        for one_of_12 in percolator_validated_list:
            one_of_12_FDR = uc.add_estimated_fdr(
                input_file = one_of_12,
            )
            one_of_12_FDR_filtered = uc.filter_csv(
                input_file = one_of_12_FDR,
            )
            twelve_filtered.append( one_of_12_FDR_filtered )

        # For the combined FDR scoring, we merge all 12 files:
        filtered_merged = uc.merge_csvs( twelve_filtered )

        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.add_estimated_fdr(
        input_file = cFDR,
    )

    # 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.filter_csv(
        input_file = cFDR_FDR,
    )
    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(
        '''
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__)
        exit()
    main(sys.argv[1])
```

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 python3.4
# 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',
    '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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/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 = []
    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,  # 'LTQ XL low res' profile!
        params = params
    )
    # 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_to_mgf_and_update_rt_lookup(
            input_file = spec_file
        )
        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' ] = ''

            if filter_before_validation == True:
                uc.params['csv_filter_rules'] = [
                    ['Modifications','contains','{0}'.format(cascade[level][1].split(
↪',')[3])]
                ]
                filtered_search_results = uc.filter_csv(
```

```
                    input_file = unified_search_results,
                )
            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.merge_csvs(
            input_files = sorted(validated_results),
            force      = force,
        )
        uc.params['csv_filter_rules'] = [
            ['Is decoy','equals','false'],
            ['PEP','lte',0.01],
        ]
        filtered_validated_results = uc.filter_csv(
                input_file = validated_results_from_all_engines,
                )
        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,
            #  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.merge_csvs(
        input_files = sorted(result_files),
    )
    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']
                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
            }
            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 python3.4
# 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'
    ),
    '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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/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 = []
    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,  # 'LTQ XL low res' profile!
        params = params
    )

    # 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.merge_csvs(
                input_files = sorted(validated_results),
            )
        filtered_validated_results = uc.filter_csv(
            input_file = validated_results_from_all_engines,
            )
        result_files.append(filtered_validated_results)

    results_all_files = uc.merge_csvs(
            input_files = sorted(result_files),
        )
    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
                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

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 python3.4
# 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' : [
        '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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/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 = []
    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,  # 'LTQ XL low res' profile!
        params = params
    )

    # 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):
                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.filter_csv(
                    input_file = unified_search_results,
                    )
                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.merge_csvs(
                input_files = sorted(validated_results),
            )
        uc.params['prefix'] = ''
        uc.params['csv_filter_rules'] = [
                ['Is decoy','equals','false'],
                ['PEP','lte',0.01],
            ]
        filtered_validated_results = uc.filter_csv(
            input_file = validated_results_from_all_engines
            )
        result_files.append(filtered_validated_results)

    results_all_files = uc.merge_csvs(
            input_files = sorted(result_files),
        )
    return results_all_files

def 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)
    '''
    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
                    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)
    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.

ap-proach	count_type	valida-tion_engine	un-modi-fied	multi-modified	Oxi-dation	Deami-dated	Methyl	Acetyl	Phos-pho	to-tal
un-grouped	psm	qvality	18813	51	332	336	102	292	122	20048
grouped	psm	qvality	21370		383	171	100	435	142	22601
cas-cade	psm	qvality	19065		424	251	111	417	38	20406
un-grouped	upep	qvality	6707	39	143	173	27	123	58	7270
grouped	upep	qvality	7525		161	104	31	191	58	8070
cas-cade	upep	qvality	7784		172	152	43	171	55	8377

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 python3.4
# 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


    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)
            exit()

    # mods from Wen et al. (2015):
    modifications = [
        'C,fix,any,Carbamidomethyl',  # Carbamidomethyl  (C) was set as fixed␣
↪modification
        'M,opt,any,Oxidation',        # Oxidation (M) as well as
        'N,opt,any,Deamidated',       # Deamidated (NQ) were set as optional␣
↪modification
        'Q,opt,any,Deamidated',       # Deamidated (NQ) were set as optional␣
↪modification
    ]


    # 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.generate_target_decoy(
        input_file = target_database,
    )

    # 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
        '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_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',       # precursor ion mass tolerance␣
↪was set to 10 ppm
                                                      # 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
        'precursor_min_charge'          : '1',        # the minimum precursor charge␣
↪to consider if charges are not specified in the spectrum file was set as 1
        'precursor_max_charge'          : '8',        # the maximum precursor charge␣
↪to consider was set as 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_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',       # precursor ion mass tolerance␣
↪was set to 10 ppm
        'frag_mass_tolerance'           : '0.6',      # the fragment ion mass␣
↪tolerance was set to 0.6 Da
        'frag_mass_tolerance_unit'      : 'da',       # the fragment ion mass␣
↪tolerance was set to 0.6 Da
        'precursor_isotope_range'       : '0,1',      # parent monoisotopic mass␣
↪isotope error was set as 'yes'
        'precursor_max_charge'          : '8',        # maximum parent charge of␣
↪spectrum was set as 8
        'enzyme'                        : 'trypsin',  # the enzyme was set as trypsin␣
↪([RK]|[X])
        'maximum_missed_cleavages'      : '1',        # the maximum missed cleavage␣
↪sites were set as 1
        # (used by default)                           # no model refinement was␣
↪employed.
```

```
        'database'                      : target_decoy_database,
        'modifications'                 : modifications,
    }


    search_engine_settings = [
        ('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'),    # not used in␣
↪Wen et al., so we use the same settings as xtandem
        ('msgfplus_v9979', msgf_params, 'LTQ XL high res'),          # the instrument␣
↪selected was 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.merge_csvs(
                input_files = percolator_validated_results,
        )
        merged_unvalidated_csv = uc.merge_csvs(
                input_files = unified_results,
        )

        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',
    )

    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__)
        exit()
    main(sys.argv[1])
```

Mgf conversion loop examples

mgf_conversion_inside_and_outside_loop.**main**()

```
#!/usr/bin/env python3.4
# 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'
    )
    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_to_mgf_and_update_rt_lookup(
        input_file       = mzML_file, # 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'
        )
        output_files.append( output_file )


    # 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()
```

Example search with search for labeling with 15N 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 python3.4
# 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. 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': 'http://www.uni-muenster.de/Biologie.IBBP.AGFufezan/misc/
↪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'
        )

    mgf_file = uc.convert_to_mgf_and_update_rt_lookup(
        input_file       = mzML_file,
    )
    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:
            unified_result_file = uc.search(
                input_file = mgf_file,
                engine     = engine,
            )

            validated_file = uc.validate(
                input_file = unified_result_file,
                engine     = 'percolator_2_08',
            )

            filtered_file = uc.filter_csv(
```

```
                input_file = validated_file,
            )

            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',
        )
    uc.params['prefix']                   = None
    uc.params['label']                    = ''
    uc.params['visualization_label_list'] = label_list
    label_comparison_file_list = []
    for label in label_list:

        label_comparison_file_list.append(
            uc.merge_csvs( files_2_merge[label] )
        )
    uc.visualize(
        input_files     = label_comparison_file_list,
        engine          = 'venndiagram',
    )

    return

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

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

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

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

CHAPTER 8

Changelog

Record of released Ursgal versions with all notable changes

Changelog

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)

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.

Version 0.3.4 (02.2016)

1. Implementation of de novo search engines: Novor, PepNovo
2. X!Tandem version Vengeance included

CHAPTER 9

FAQ

Frequently Asked Questions

Installation

Q: MS Amanda does not work on Unix. What could be the problem ?

A: To run MSAmanda one needs to install the Mono frameweork. Visit http://www.mono-project.com/ for proper installation instructions.

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 :)

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$ python3.4 setup.py install
```

CHAPTER 10

Known Issues

Known Issues

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

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

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.

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