ursgal Documentation

Release ursgal_version = '0.6.2'

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

Nov 02, 2018
# Contents

## 1 Introduction
- 1.1 Update to v0.6.0 Warning ................................................. 3
- 1.2 Summary .................................................................................. 3
- 1.3 Abstract .................................................................................... 4
- 1.4 Download .................................................................................. 4
- 1.5 Installation ................................................................................ 4
- 1.6 Tests .......................................................................................... 5
- 1.7 Questions and Participation ....................................................... 5
- 1.8 Documentation .......................................................................... 6
- 1.9 Disclaimer .................................................................................. 6
- 1.10 Copyrights ............................................................................... 6
- 1.11 Contact ..................................................................................... 6
- 1.12 Citation ................................................................................... 6

## 2 Quick Start
- 2.1 Quick Start Tutorial ................................................................. 9

## 3 Module structure
- 3.1 General Structure ...................................................................... 13

## 4 Urganal module contents
- 4.1 UController ............................................................................. 23
- 4.2 UNode ....................................................................................... 36
- 4.3 UCore ....................................................................................... 42
- 4.4 UMapMaster ........................................................................... 43
- 4.5 Chemical Composition ........................................................... 44
- 4.6 Unimod Mapper ....................................................................... 46

## 5 Engines
- 5.1 Included engines ..................................................................... 51
- 5.2 How to extend and create new engines ? ................................. 78

## 6 Parameter

## 7 Examples
- 7.1 Example Scripts ....................................................................... 89
The latest Documentation was generated on: Nov 02, 2018
1.1 Update to v0.6.0 Warning

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

1.2 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 eight different search engines (some available in multiple versions), including three open modification search engines
2. Evaluation and post processing of search results with up to two different engines
3. Integration of search results from different search engines
4. De novo sequencing with up to two different search engines
5. Miscellaneous tools including the creation of a target decoy database as well as filtering, sanitizing and visualizing of results
1.3 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 (qality (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.


1.4 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/

1.5 Installation

Ursgal requires Python 3.4 or higher.

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

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

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

Install third party engines:

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

**Note:** Since we are not allowed to distribute all third party engines, you might need to download and install them on your own. See FAQ (*Q: How do I add an engine that is not installed via install_resources.py?*) and the respective engine documentation for more information.

Install Urüsgal:

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.)

### 1.6 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.

### 1.7 Questions and Participation

If you encounter any problems you can open up issues at GitHub, join the conversation at Gitter, or write an email to ursgal.team@gmail.com. Please also check the [Frequently Asked Questions](https://github.com/ursgal/ursgal). For any contributions, fork us at https://github.com/ursgal/ursgal and open up pull requests! Please also check the [Contribution Guidelines](https://github.com/ursgal/ursgal). Thanks!
1.8 Documentation

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

1.9 Disclaimer

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

1.10 Copyrights

Copyright 2014-2018 by authors and contributors in alphabetical order

- Christian Fufezan
- Aime B. Igiraneza
- Manuel Koesters
- Lukas P. M. Kremer
- Johannes Leufken
- Purevdulam Oyunchimeg
- Stefan Schulze
- Fengchao Yu
- Lukas Vaut

1.11 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

1.12 Citation

Ursgal citation

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

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


2.1 Quick Start Tutorial

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

2.1.1 1. Getting Started

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

```python
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:

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

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

2.1.2 2. Running a Simple Search

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

- a peptide database (.fasta) that will be searched
- one or more mass spectrometer output files (.mzML or .mgf)
Once you have these files, you are ready to execute a full search with Ursgal:

```python
import ursgal
uc = ursgal.UController(
    params = {'database': 'my_database.fasta'}
)
search_result = uc.search(
    input_file = 'my_mass_spec_file.mzML',
    engine = 'omssa',
)
```

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

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

### 2.1.3 3. Adjusting Parameters

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

```python
# 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:

```python
uc = ursgal.UController(
    params = {'database': 'my_database.fasta'},
    profile = 'QExactive+'
)
```
2.1.4 4. Available Workflow Functions

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

```python
search_result = uc.search(
    input_file = 'my_mass_spec_file.mzML',
    engine = 'omssa',
)
validated_result = uc.validate(
    input_file = search_result,
    engine = 'percolator_2_08',
)
```

Currently, the following UController workflow functions are available:

- `UController.combine_search_results()` Statistical integration of search results from multiple engines
- `UController.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 to calculate the false-discovery rate (FDR, q-value) and/or posterior error probability (PEP)
- `UController.visualize()` Visualization of results

2.1.5 5. Building Custom workflows

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

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

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

```python
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):

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

### 2.1.6 6. JSONs, Force and File Names

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

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

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

### 2.1.7 7. Example Scripts

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

These example scripts are a good starting point:

- `simple_example_search.py`
- `target_decoy_generation_example.py`
- `do_it_all_folder_wide.py`
Module structure

3.1 General Structure

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

3.1.1 Schematic Overview

3.1.2 Resources

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

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

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

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

1. <installation path of ursgal>/resources/<platform>/architectures> 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.
Chapter 3. Module structure
3.1.3 Wrapper Python class

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

The default structure of a wrapper is:

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

class xtandem_alanine( ursgal.UNode ):
    '''
    X!Tandem UNode
    Parameter options at http://www.thegpm.org/TANDEM/api/
    '''
    META_INFO = { ... }

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

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

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

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

The META INFO

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

Edit Version

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

```python
META_INFO = {
    ... 
    'edit_version' : 1.00, 
    ... 
}
```
Name, Version and Release Date

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

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

Engine Type

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

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

Citation

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

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

Input Extensions

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

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

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

Create own folder

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

```python
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.

```python
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 :)

3.1. General Structure
Distributable

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

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.

Download information

The download information is required for the install_resources.py script to function.
3.1.4 Grouped parameters - uparams.py

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

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

Entries:

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

• ‘uvalue_type’ Defines the uvalue type of this parameter.

• ‘uvalue_option’ Provides informations for parameter settings in the GU, e.g. possible parameter value ranges and step sizes. The required informations depend on the uvalue type and are listed in the following.

uvalue_option entries for different uvalue_types:

• ‘bool’: no uvalue_options are required

• ‘float’/‘int’:
  ‘none_val’: This value will be set to None (since None cannot be entered in the GUI)
  ‘max’: maximal value to be entered
  ‘min’: minimal value to be entered
  ‘f-point’: number of decimal points (not available/needed for uvalue_type ‘int’)
  ‘updownval’: step size of user control arrows/slider
  ‘unit’: unit of the uvalue

• ‘str’:
  ‘none_val’: This value will be set to None (since None cannot be entered in the GUI)
  ‘multiple_line’: True, if the user should be allowed to enter multiple lines

• ‘list’:
  ‘none_val’: This value will be set to None (since None cannot be entered in the GUI)
  ‘item_title’: generic title for the items in the list
  ‘item_type’: type of items in the list,
  ‘custom_val_max’: maximum number of entries (?)

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

• ‘dict’:
  ‘none_val’: This value will be set to None (since None cannot be entered in the GUI)
  ‘multiple_line’: True, if the user should be allowed to enter multiple lines (given for each key in the dict)
  ‘item_titles’: generic titles for the keys and values in the dict (given in the same structure as the dict)
  ‘value_types’: types of values for each key in the dict

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

• ‘select’:
  ‘select_type’: type of selection from available_values, e.g. ‘combo_box’ if combinations of multiple values can be selected, ‘radio_button’ if only one value can be selected
  ‘available_values’: list of available values to be selected
  ‘custom_val_max’: maximum number of entries (?)

The following example shows the parameter dict for the ‘frag_mass_tolerance’ parameter.
ursgal_params = {
    'frag_mass_tolerance': {
        'edit_version': 1.00,
        'available_in_unode': [
            'moda_v1_51',
            'msamanda_1_0_0_5242',
            'msamanda_1_0_0_5243',
            'msamanda_1_0_0_6299',
            'msamanda_1_0_0_6300',
            'msamanda_1_0_0_7503',
            'msamanda_1_0_0_7504',
            'msfragger_20170103',
            'myrimatch_2_1_138',
            'myrimatch_2_2_140',
            'novor_1_{beta}',
            'omssa_2_1_9',
            'pepnovo_3_1',
            'pipi_1_4_5',
            'pipi_1_4_6',
            'xtandem_cyclone_2010',
            'xtandem_jackhammer',
            'xtandem_piledriver',
            'xtandem_sledgehammer',
            'xtandem_vengeance',
            'xtandem_alanine',
        ],
        'triggers_rerun': True,
        'ukey_translation': {
            'moda_style_1': 'FragTolerance',
            'msamanda_style_1': 'ms2_tol',
            'msfragger_style_1': 'fragment_mass_tolerance',
            'myrimatch_style_1': 'FragmentMzTolerance',
            'novor_style_1': 'fragmentIonErrorTol',
            'omssa_style_1': '-to',
            'pepnovo_style_1': '-fragment_tolerance',
            'pipi_style_1': 'ms2_tolerance',
            'xtandem_style_1': 'spectrum, fragment monoisotopic mass error',
        },
        'utag': [
            'fragment',
        ],
        'uvalue_translation': {},
        'uvalue_type': 'int',
        'uvalue_option': {
            'none_val': None,
            'max': 100000,
            'min': 0,
            'updownval': 1,
            'unit': ''
        },
        'default_value': 20,
        'description': 'Mass tolerance of measured and calculated fragment ions',
    },
}
4.1 UController

```python
class ursgal.ucontroller.UController(*args, **kwargs)
    ursgal main class

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

    Currently available profiles are:
      – 'QExactive+'
      – 'LTQ XL high res'
      – 'LTQ XL low res'

    Example:

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

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

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

    Keyword Arguments
    • input_files (list) – A list containing the complete paths to two or more input files.
      Input files have to be unified result `.csv` files that were produced by different engines.
• **engine** (*str*) – The name of the desired search result combiner. Can also be a shortened version if it is unambiguous.

• **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:

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

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

**Returns**  Path of the output file

**Return type**  *str*

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

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

**Keyword Arguments**

• **input_file** (*str*) – The complete path to the input file.

• **engine** (*str*) – The name of the desired converter engine. Can also be a shortened version if it is unambiguous.

• **force** (*bool*) – (Re)do the analysis, even if output file already exists.

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

• **guess_engine** (*bool*) – The converter engine is guessed based on the input file. This works so far for mzml2mgf conversion and conversion of search_engine result files to csv.

Example:

```python
>>> 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:

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

Returns Path of the output file

Return type str

Notes: internal function, use convert() instead

convert_to_mgf_and_update_rt_lookup(input_file, force=None, output_file_name=None)

Converts the mzML to mgf and updates the scanID to retention time lookup. The lookup is needed for the unifying of the .csv files.

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

Returns name of the output mgf file

Return type str

Notes: internal function, use convert() instead

determine_availability_of_unodes()

The ucontroller determine_availability_of_unodes function

Note: internal function

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

distinguish_multi_and_single_input(in_input)

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

download_resources(resources=None)

Function to download all executable from the specified http url

Keyword Arguments **resources** (*list*) – list of specific resources that should be downloaded. 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.
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 input will yield omssa_2_1_9 if there is only one omssa engine installed, i.e. the mapping (<stored_full_engine_name>.startswith(<input>) has to be unique and defined.

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

Note: internal function, since assertion error is called.

**Parameters**

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

calls `self.guess_engine_name()`

**Returns** Full name of the engine or None.

**Return type** *str*

eval_if_run_needs_to_be_executed *(engine=None, force=None)*

Returns the reason why self.run needs to be executed or None if there is no need

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

The UController execute_misc_engine function

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

**Keyword Arguments**

* **input_file** *(str)* – The complete path to the input, a unified (and possibly merged) search result .csv.

* **engine** *(str)* – the name of the validation engine which should be run, can also be a short version if this name is unambiguous

* **force** *(bool)* – (Re)do the analysis, even if output file already exists.

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

* **merge_duplicates** *(bool)* – If True, the produced output file will be checked for duplicated PSMs, which will be merged into a single line. Caution, the original output file will be overwritten!

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

Example:

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

**Returns** Path of the output file

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

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

**Keyword Arguments**

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

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

**fetch_file** *(engine=None)*

The UController `fetch_file` function

Downloads files (FTP or HTTP).

**Keyword Arguments**

- **engine** *(str)* – Available options are ‘get_http_files_1_0_0’ and ‘get_ftp_files_1_0_0’

**Example:**

```python
>>> 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'],
    ...     'ftp_output_folder' : '/home/Desktop/',
    ...
}
>>> uc = ursgal.UController(
    ...     params = params
    ...
)
>>> uc.fetch_file(
    ...     engine = 'get_ftp_files_1_0_0'
    ...
)
```

**Returns** Path of the downloaded file

**Return type** *str*

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

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

The UController `filter_csv` function

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

**Keyword Arguments**

- **input_file** *(str)* – The complete path to the input, input file has currently to be a .csv file.
• **force (bool)** – (Re)do the analysis, even if output file already exists.

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

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

### Example

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

### generate_multi_file_dicts (input_files)

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

### generate_multi_helper_file (input_files)

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

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

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

The ucontroller function for target_decoy database generation.

**Keyword Arguments**

• **input_files (list)** – List with complete paths to one or more fasta databases.

• **engine (str)** – Name of the database generator which should be run, can also be a short version if this name is unambiguous.

• **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:**

```python
>>> 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:**
```python
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 ambiguous name or if a engine has multiple version, it is required to name the engine unambiguously. Instead of myrimatch use myrimatch_2_1_138.

Parameters short_engine (str) – engine short name or tag

Notes: internal function

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

Return type str

input_file_sanity_check(input_file, engine=None, extensions=None, multi=False, custom_str=None)
The ucontroller input_file_sanity_check function Asserts that input files exist, can be read, have the right file type and file extension etc. Raises an AssertionError if any criterion is violated.

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

Note: Internal Function

Returns None

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

[ WARNING ] This function is not supported anymore! Please use execute_misc_engine() instead
The ucontroller function to call the upeptide_mapper node.

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

**Available converter nodes**

- upeptide_mapper_1_0_0

**Available converter classes of upeptide_mapper_1_0_0**

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

**Keyword Arguments**

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

**Returns** Path of the output file

**Return type** str

**merge_csvs**

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:**

```python
>>> 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)

print_old_function_warning(old_funct_name, new_func_name)

quantify(input_file, engine, force=None, output_file_name=None, multi=False)

The ucontroller quantify function

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

Keyword Arguments

- **input_file** (str) – The complete path to the mzML file.
- **engine** (str) – The name of the quantification engine which should be used, can also
  be a short version if this name is unambiguous.
- **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:

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

Returns Path of the output file (unified CSV format)

Return type  str

run_unode_if_required(force, engine_name, answer, merge_duplicates=False, history_addon=None)

The ucontroller run_unode_if_required function

Note: internal function

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

Keyword Arguments

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

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

search(input_file, engine=None, force=None, output_file_name=None)

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 unambiguous.

• force (bool) – (Re)do the analysis, even if output file already exists.

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

Example::

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

>>> uc.search(...
...     input_file = 'BSA.mzML',
...     engine = 'omssa',
...)

Returns Path of the output file (unified CSV format)

Return type str

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

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

• convert() (mzml to mgf, if required, using the mzml2mgf engine)

• search_mgf()

• convert() (raw search results to csv, if required)

• execute_misc_engine() (peptide_mapper)

• execute_misc_engine() (unify_csv)

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

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 Usgal)
- **engine**(str) – the name of the identification engine which should be run, can also be a short version if this name is unambiguous.
- **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:**

```python
>>> 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 specified 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:**
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 availability and installation is verified.

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

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

The ucontroller unify_csv function

Unifies the .csv files which were converted by the mzidentml library. The corrections for each engine are listed in the node under ursgal/resources/arc_independent/unify_csv_1.0.0

**Keyword Arguments**

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

- **force** *(bool)* – (Re)do the analysis, even if output file already exists.

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

**Example:**

```python
>>> 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=None, engine=None, force=None, output_file_name=None)
The UController validate function

Does statistical post-processing of unified search result .csv files with the specified validation engine.
Depending on the validation method a posterior error probability (PEP) and/or a q-value will be available
in the final results.

Keyword Arguments

• input_file (str) – The complete path to the input, a unified (and possibly merged)
search result .csv.

• engine (str) – the name of the validation engine which should be run, can also be a
short version if this name is unambiguous

• force (bool) – (Re)do the analysis, even if output file already exists.

• output_file_name (str or None) – Desired output file name excluding path (op-
tional). 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:

```python
>>> 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 pro-
duced or not to throw a proper exception in case the engine crashed.

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

Does graphical visualization of result .csv files.

Keyword Arguments

• input_files (list) – list with complete paths of .csv files

• engine (str) – the name of the visualizer which should be run, can also be a short
version if this name is unambiguous
• **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:

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

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

**Returns** Path of the output file

**Return type** `str`

### 4.2 UNode

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

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

**__init__(args, **kwargs)*

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

**__weakref__**
- list of weak references to the object (if defined)

**_execute()**
- The _execute unode function

Executes the unode executable via shell.

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

**Returns** None

```python
_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


**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 multiple engines have been used, this can be set to True. Then reports a list of used engines.

**Examples**

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

**Returns**  
The name of the last engine that was used.

**Return type**  
`str`

`get_last_engine_type (history=None)`

The unode `get_last_engine_type` function

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

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

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

**Return type** str

### get_last_search_engine (history=None, multiple_engines=False)

The uinode 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 multiple engines have been used, this can be set to True. Then reports a list of used engines.

Examples

```python
>>> 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 uinode 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

```python
>>> 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

```python
>>> [
...   "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 ambiguous, 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 ambiguous for some databases.
If the peptide occurs several times in the protein, all occurrences 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 sequences 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

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

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

```python
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.
- `force` *(#)* – (re)do the analysis if output files already exists


**Return type** dict

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

```python
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.

```python
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.

```python
update_params_with_io_data()
```
Generates a flat structure in params combining `io['input']['finfo']` & `io['output']['finfo']`
4.3 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_dalton_to_ppm(da_value, base_mz=1000.0)
Convert the precision in Dalton to ppm

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

Returns value in ppm
Return type float

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

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

Returns value in Dalton
Return type float

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

ursgal.ucore.digest(sequence, enzyme, 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 digests peptides
Return type list

ursgal.ucore.merge_duplicate_psm_rows(csv_file_path=None, psm_counter=None, psm_defining_colnames=None, joinchar='<|>', overwrite_file=True)
Rows describing the same PSM (e.g. when two proteins share the same peptide) are merged to one row.
ursgal.ucore.merge_rowdicts(list_of_rowdicts, joinchar='<|>')
Merges CSV rows. If the column values are conflicting, they are joined with a character (joinchar).

ursgal.ucore.parse_fasta(io)
Small function to efficiently parse a file in fasta format.

Keyword Arguments:

- **io**(obj) – openend file obj (fasta file)

Yields:

- tuple – fasta_id and sequence

ursgal.ucore.print_current_params(params, old_params=None)
Function to print current params

Keyword Arguments:

- **params**(dict) – parameter dict to print

ursgal.ucore.reformat_peptide(regex_pattern, unimod_name, peptide)
reformats the MQ and Novor peptide string to ursgal format (ac)SSSLM(ox)RPGPSR → SSSLMRPG-PSR#Acetyl:0;Oxidation:5

ursgal.ucore.terminal_supports_color()
Returns True if the running system’s terminal supports color, and False otherwise. Source: https://github.com/django/django/blob/master/django/core/management/color.py

### 4.4 UMapMaster

Mapping classes of Ursgal

#### 4.4.1 UPParamMapper

class ursgal.umapmaster.UPParamMapper(*args, **kwargs)
UPParamMapper class offers interface to ursgal.uparams

By default, the ursgal.uparams are parsed and the UPParamMapper 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:

```python
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_sledgehammer',
            'xtandem_cylone',
            ...
        ],
        'omssa_style_1' ...
    },
    # This is done during uNode initializations
    # each uNode will register its style with umapmaster
    #
    'engine_2_style': {
        'xtandem_sledgehammer': 'xtandem_style_1', ...
    },
    'engine_2_params': {
        'xtandem_sledgehammer': [ 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

4.5 Chemical Composition

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

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

Keyword Arguments

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

• **aa_compositions** *(Optional[dict])* – amino acid compositions

• **isotopic_distributions** *(Optional[dict])* – isotopic distributions

Keyword argument examples:

    sequence - Currently this can for example be:: [  `+H2O2H2-OH`, `+{0}`.format(`H2O`),
    `[peptide]`.format(peptide=`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'

(continues on next page)
>>> c.hill_notation_unimod()
'C(52)H(90)N(10)O(18)'

>>> c
{'O': 18, 'H': 90, 'C': 52, 'N': 10}

>>> c.composition_of_mod_at_pos[1]
defaultdict(<class 'int'>, {'O': 1, 'H': 2, 'C': 2})

>>> c.composition_of_aa_at_pos[1]
{'O': 3, 'H': 7, 'C': 5, 'N': 1}

>>> c.composition_at_pos[1]
defaultdict(<class 'int'>, {'O': 4, 'H': 9, 'C': 7, 'N': 1})

>>> c = ursgal.ChemicalComposition('+H2O2H2')

>>> c
{'O': 2, 'H': 4}

>>> c.subtract_chemical_formula('H3')

>>> c
{'O': 2, 'H': 1}

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

add_chemical_formula(chemical_formula, factor=1)
Adds chemical formula to the instance

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

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

add_glycan(glycan)
Adds a glycan to the instance.

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

add_peptide(peptide)
Adds peptide sequence to the instance

clear()
Resets all lookup dictionaries and self

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

composition_at_pos = None
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.

```python
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.

```python
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*

```python
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*

```python
subtract_chemical_formula(chemical_formula, factor=1)
```

Subtract chemical formula from instance

**Parameters**

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

**Keyword Arguments**

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

```python
subtract_peptide(peptide)
```

Subtract peptide from instance

```python
use(sequence)
```

Re-initialize the class with a new sequence

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

**Parameters**

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

### 4.6 Unimod Mapper

```python
class ursgal.UnimodMapper
```

UnimodMapper class that creates lookup to the unimod.xml and user-defined_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 Uni-
mod is compared to the input, i.e. round( mass, decimal_places )

Returns Dicts of elements
Return type list

Examples::

```python
>>> import ursgal
>>> U = ursgal.UnimodMapper()
>>> U.appMass2element_list(18, decimal_places=0)
 {‘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 Uni-
mod is compared to the input, i.e. round( mass, decimal_places )

Returns Unimod IDs
Return type list

Examples::

```python
>>> import ursgal
>>> U = ursgal.UnimodMapper()
>>> U.appMass2id_list(18, decimal_places=0)
```

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 Uni-
mod is compared to the input, i.e. round( mass, decimal_places )

Returns Unimod names
Return type list

Examples::

```python
>>> 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 multiple 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 multiple 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 multiple 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 multiple 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 multiple 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 unimod ID

Parameters unimod\_name \( (str) \)

Returns Unimod id

Return type int

name2mass \( (unimod\_name) \)

Converts unimod name to unimod mono isotopic mass

Parameters unimod\_name \( (str) \)

Returns Unimod mono isotopic mass

Return type float

writeXML \( (modification\_dict, xmlFile=None) \)

Writes a unimod-style userdefined_unimod.xml file in ursal/resources/platform_independent/arc_independent/ext

Parameters

- modification\_dict \( (dict) \) – dictionary containing at least
- 'mass' (mass of the modification) –
- 'name' (name of the modification) –
- 'composition' (chemical composition of the modification as a Hill notation) –
Ursgal allows existing programs to be incorporated with ease. We call those programs or scripts engines. Currently, ursgal comes with these engines:

## 5.1 Included engines

### 5.1.1 Available Search Engines

**Protein Database Search Engines**

**MS Amanda**

Available MS Amanda versions, starting with the newest version:

```python
class ursgal.wrappers.msamanda_2_0_0_10695.msamanda_2_0_0_10695(*args, **kwargs)
    MS Amanda 2_0_0_9706 UNode
    Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_9706.msamanda_2_0_0_9706(*args, **kwargs)
    MS Amanda 2_0_0_9706 UNode
    Import functions from msamanda_2_0_0_9695

class ursgal.wrappers.msamanda_2_0_0_9695.msamanda_2_0_0_9695(*args, **kwargs)
    MS Amanda 2_0_0_9695 UNode Parameter options at http://ms.imp.ac.at/inc/pd-nodes/msamanda/Manual%20MS%20Amanda%20Standalone.pdf
```

Note: Please download and install MS Amanda manually from http://ms.imp.ac.at/?goto=msamanda

postflight()
    Convert .tsv result files to .csv

preflight()
    Formatting the command line via self.params
    Settings file is created in the output folder and added to self.created_tmp_files (can be deleted)

    Returns  self.params(dict)

class ursgal.wrappers.msamanda_1.0.0.7504.msamanda_1.0.0.7504(*args, **kwargs)
    MSAmanda 1.0.0.7504 UNode
    Import functions from msamanda_1.0.0.5243

class ursgal.wrappers.msamanda_1.0.0.7503.msamanda_1.0.0.7503(*args, **kwargs)
    MSAmanda 1.0.0.7503 UNode
    Import functions from msamanda_1.0.0.5243

class ursgal.wrappers.msamanda_1.0.0.5243.msamanda_1.0.0.5243(*args, **kwargs)
    MSAmanda 1.0.0.5243 UNode Parameter options at http://ms.imp.ac.at/inc/pd-nodes/msamanda/Manual%20MS%20Amanda%20Standalone.pdf

postflight()
    Convert .tsv result files to .csv

preflight()
    Formatting the command line via self.params
    Settings file is created in the output folder and added to self.created_tmp_files (can be deleted)

    Returns  self.params(dict)

class ursgal.wrappers.msamanda_1.0.0.5242.msamanda_1.0.0.5242(*args, **kwargs)
    MSAmanda 1.0.0.5242 UNode
    Import functions from msamanda_1.0.0.5243

MS-GF+

Available MS-GF+ versions, starting with the newest version:

class ursgal.wrappers.msgfplus_v2018_09_12.msgfplus_v2018_09_12(*args, **kwargs)
    MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf
    Import node for version 2016_09_16

class ursgal.wrappers.msgfplus_v2018_06_28.msgfplus_v2018_06_28(*args, **kwargs)
    MSGF+ UNode Parameter options at https://omics.pnl.gov/software/ms-gf
## MSFragger

Available MSFragger versions, starting with the newest version:
class ursgal.wrappers.msfragger_20171106.msfragger_20171106(*args, **kwargs)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html


Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

class ursgal.wrappers.msfragger_20170103.msfragger_20170103(*args, **kwargs)
MSFragger unode

Note: Please download and install MSFragger manually from http://www.nesvilab.org/software.html


Note: Addition of user amino acids not implemented yet. Only mzML search possible at the moment. The mgf file can still be passed to the node, but the mzML has to be in the same folder as the mgf.

Warning: Still in testing phase! Metabolic labeling based 15N search may still be errorprone. Use with care!

postflight()
Reads MSFragger tsv output and write final csv output file.

Adds:
  • Raw data location, since this can not be added later
  • Converts masses in Da to m/z (could be done in unify_csv)

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params

Return type dict

MODa

Available MODa versions, starting with the newest version:
class ursgal.wrappers.moda_v1_61.moda_v1_61(*args, **kwargs)
    MODa UNode Check http://prix.hanyang.ac.kr/download/moda.jsp for download, new versions and contact information


    Import functions from moda_v1_51

class ursgal.wrappers.moda_v1_51.moda_v1_51(*args, **kwargs)
    MODa UNode Check http://prix.hanyang.ac.kr/download/moda.jsp for download, new versions and contact information


    postflight()
        Rewrite ModA output .tsv into .csv so that it can be unified

    preflight()
        Formatting the command line via self.params

            Returns self.params

            Return type dict

MyriMatch

class ursgal.wrappers.myrimatch_2_1_138.myrimatch_2_1_138(*args, **kwargs)
    Myrimatch UNode


    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

OMSSA

class ursgal.wrappers.omssa_2_1_9.omssa_2_1_9(*args, **kwargs)
    omssa_2_1_9 UNode


    OMSSA 2.1.9 parameters at http://proteomicsresource.washington.edu/protocols06/omssa.php

5.1. Included engines

postflight()
Will correct the OMSSA headers and add the column retention time to the csv file

preflight()
Formatting the command line via self.params
unimod Modifications are translated to OMSSA modifications

Returns self.params(dict)

PIPI

Available PIPI versions, starting with the newest version:

class ursgal.wrappers.pipi_1_4_5.pipi_1_4_5(*args,**kwargs)
Unode for PIPI: PTM-Invariant Peptide Identification For further information see: http://bioinformatics.ust.hk/pipi.html

Note: Please download and extract PIPI manually from http://bioinformatics.ust.hk/pipi.html


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

preflight()
Formatting the command line and writing the param input file via self.params

Returns self.params
Return type dict

class ursgal.wrappers.pipi_1_4_6.pipi_1_4_6(*args,**kwargs)
Unode for PIPI: PTM-Invariant Peptide Identification For further information see: http://bioinformatics.ust.hk/pipi.html

Note: Please download and extract PIPI manually from http://bioinformatics.ust.hk/pipi.html


X!Tandem

Available X!Tandem versions, starting with the newest version:

class ursgal.wrappers.xtandem_alanine.xtandem_alanine(*args,**kwargs)
X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/

class ursgal.wrappers.xtandem_vengeance.xtandem_vengeance(*args, **kwargs)

X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/


format_templates()
Returns formatted X!Tandem input files

The formatting is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

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

preflight()
Formatting the command line via self.params

Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params

Return type dict

class ursgal.wrappers.xtandem_piledriver.xtandem_piledriver(*args, **kwargs)

X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/


format_templates()
Returns formatted X!Tandem input files

The formatting is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

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

preflight()
Formatting the command line via self.params

Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params

Return type dict

class ursgal.wrappers.xtandem_sledgehammer.xtandem_sledgehammer(*args, **kwargs)

X!Tandem UNode Parameter options at http://www.thegpm.org/TANDEM/api/


format_templates()
Returns formatted X!Tandem input files

The formatting is taken from self.params

Returns keys are the names of the three templates (15N-masses.xml, taxonomy.xml, input.xml)

Return type dict

5.1. Included engines
postflight()
This can be/is overwritten by the engine uNode class

preflight()
Formatting the command line via self.params
Input files from format_templates are created in the output folder and added to self.created_tmp_files (can be deleted)

Returns self.params
Return type dict

class ursgal.wrappers.xtandem_jackhammer.xtandem_jackhammer(*args, **kwargs)
class ursgal.wrappers.xtandem_cyclone_2010.xtandem_cyclone_2010(*args, **kwargs)

De Novo Search Engines

Novor
class ursgal.wrappers.novor_1_1beta.novor_1_1beta(*args, **kwargs)
Novor UNode Parameter options at http://rapidnovor.com/
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/
postflight()
Reformats the PepNovo output file
preflight()
Formatting the command line via self.params

Returns self.params
Return type dict
Cross Link Search Engines

Kojak

class ursgal.wrappers.kojak_1_5_3.kojak_1_5_3(*args, **kwargs)
Kojak UNode Parameter options at http://www.kojak-ms.org/param/index.html


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 parameter file is used and adjusted.

Returns keys are the names of the parameter 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

5.1.2 Converter Engines

Convert CSV to SSL 1.0.0

class ursgal.wrappers.csv2ssl_1_0_0.csv2ssl_1_0_0(*args, **kwargs)
csv2ssl_1_0_0 UNode

_execute()
Result files (.csv) are converted to spectrum sequence list (.ssl) files. These .ssl can be used as input files for BiblioSpec.

Input file has to be a .csv

Creates a _converted.csv file and returns its path.

ursgal.resources.platform_independent.arc_independent.csv2ssl_1_0_0.csv2ssl_1_0_0.main(input, output, score, score)

Convert csvs to ssl

Convert CSV to Counted Results

class ursgal.wrappers.csv2counted_results_1_0_0.csv2counted_results_1_0_0(*args, **kwargs)
csv2counted_results_1_0_0 UNode

5.1. Included engines
Results (.csv) are summarized as table (.csv) containing all identified proteins, peptides, or other specified identifiers. For each sample, the peptide or spectral count for each identifier is given.

Input file has to be a .csv

Creates a _counted.csv file and returns its path.

Columns containing the elements that should be counted (identifiers) are given as a list of headers using uc.params["identifier_column_names"]. Columns defining a unique countable element (e.g. “Sequence”, “Spectrum ID”) are given as a list of headers using uc.params["count_column_names”].

This can be used to create a SFINX (http://sfinx.ugent.be/) input file, using:

uc.params["convert_to_sfinx"] = True
uc.params["identifier_column_names"] = ["Protein ID"]
uc.params["count_column_names"] = ["Sequence"]

Keyword Arguments

• **input_file (str)** – name including path for the input file

• **output_file (str)** – name including path for the output file

• **identifier_column_names (list)** – list of column headers that define the identifier. Multiple column names are joined for combined identifiers.

• **count_column_names (list)** – list of column headers which are used for counting.

• **count_by_file (bool)** – the number of unique hits for each identifier is given in separate columns for each raw file (file name as defined in Spectrum Title)

• **convert2sfinx (bool)** – If True, the header of the identifier column is “rownames”. If False, the joined header name will be used.

• **keep_column_names (list)** – list of column headers which are not used as identifiers but kept in the output, e.g. when counting ["Sequence", "Modifications"] the column ["Protein ID"] could be specified here. Multiple entries for one identifier (e.g. when identifier_column_names = ["Protein ID"] and keep_column_names = ["Sequence"]) are seperated by ‘<#>’.
Convert Mascot DAT to CSV

class ursgal.wrappers.mascot_dat2csv_1_0_0.mascot_dat2csv_1_0_0(*args, **kwargs)

  Dummy to merge mascot data into usgal workflow

Convert MS-GF+ MZID to CSV

class ursgal.wrappers.msgfplus2csv_py_v1_0_0.msgfplus2csv_py_v1_0_0(*args, **kwargs)

msgfplus2csv_py v1.0.0 UNode

class ursgal.wrappers.msgfplus2csv_v1_2_1.msgfplus2csv_v1_2_1(*args, **kwargs)

msgfplus2csv_v1.2.1 UNode Parameter options at https://omics.pnl.gov/software/ms-gf


postflight()
  Convert .tsv result file to .csv and translates headers

preflight()
  mzid result files from MS-GF+ are converted to CSV using the MzIDToTsv converter from MS-GF+
  Input file has to be a .mzid or .mzid.gz
  Creates a .csv file and returns its path
  Mzid to Tsv Converter Usage: MzIDToTsvConverter -mzid:"mzid path" [-tsv:"tsv output path"] [-unroll-]
  Required parameters: ‘-mzid:path’ - path to mzid.gz file; if path has spaces, it must be in quotes.
  Optional parameters: ‘-tsv:path’ - path to tsv file to be written; if not specified, will be output
  to same location as mzid ‘-unroll-u’ signifies that results should be unrolled - one line per
  unique peptide/protein combination in each spectrum identification ‘-showDecoy-sd’ signifies that
decoy results should be included in the result tsv


class ursgal.wrappers.msgfplus2csv_v2017_07_04.msgfplus2csv_v2017_07_04(*args, **kwargs)

msgfplus_C_mzid2csv_v2017_07_04 UNode Parameter options at https://omics.pnl.gov/software/ms-gf


postflight()
  Convert .tsv result file to .csv and translates headers

preflight()
  mzid result files from MS-GF+ are converted to CSV using the MzIDToTsv converter from MS-GF+
  Input file has to be a .mzid or .mzid.gz

5.1. Included engines
Creates a .csv file and returns its path

Mzid to Tsv Converter Usage: MzidToTsvConverter -mzid:"mzid path" [-tsv:"tsv output path"] [-unroll-u] [-showDecoy-sd]

**Required parameters:** `-mzid:path` - path to mzid.gz file; if path has spaces, it must be in quotes.

**Optional parameters:** `-tsv:path` - path to tsv file to be written; if not specified, will be output to same location as mzid `-unroll-u` signifies that results should be unrolled - one line per unique peptide/protein combination in each spectrum identification `-showDecoy-sd` signifies that decoy results should be included in the result tsv

```python
class ursgal.wrappers.msgfplus2csv_v2017_01_27.msgfplus2csv_v2017_01_27(*args, **kwargs)
msgfplus2csv_v2017_01_27 UNode Parameter options at https://omics.pnl.gov/software/ms-gf
Reference:
```

```python
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
```

**postflight**
Convert .tsv result file to .csv

**preflight**
Convert .mzML files into .mgf files

**Convert MZML to MGF 1.0.0**

```python
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**

```python
class ursgal.wrappers.xtandem2csv_1_0_0.xtandem2csv_1_0_0(*args, **kwargs)
xtandem2csv_1_0_0 UNode
```

62 Chapter 5. Engines

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 occurrence of a peptide in the same protein) are not reported.

**MzidLib**

```python
class ursgal.wrappers.mzidentml_lib_1_7.mzidentml_lib_1_7(*args, **kwargs)
    MzidLib 1_7 UNode
    Import functions from mzidentml_lib_1_6_10

Note: Please download and install manually from http://www.proteoannotator.org/?q=installation
```

```python
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


    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
```

**Unify CSV v1.0.0**

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

    Unifies the .csv files of converted search engine results. The corrections for each engine are listed in the node under ursgal/resources/platform_independent/arc_independent/unify_csv_1_0_0

    _execute()
        Result files from search engines are unified to contain the same informations in the same style
        Input file has to be a .csv
        Creates a _unified.csv file and returns its path
```

5.1. Included engines
5.1.3 Other Engines

Fetcher

Get FTP Files 1.0.0

class ursgal.wrappers.get_ftp_files_1_0_0.get_ftp_files_1_0_0(*args, **kwargs)

get_ftp_files_1_0_0 UNode

Downloads files from FTP servers

Note: meta info param ‘output_extensions’ is by default txt, so that the temporary txt json files get properly deleted

_parameters_

• ftp_url (*) –
• folder (*) –
• login (*) –
• password (*) –
• include_ext (*) –
• output_folder (*) –
• max_number_of_files (*) –
• blocksize (*) –

__execute__()

Downloads files from FTP server

_get_http_files_1_0_0

class ursgal.wrappers.get_http_files_1_0_0.get_http_files_1_0_0(*args, **kwargs)

_get_http_files_1_0_0 UNode

Downloads files via http
Parameters

- `http_url` (*)
- `http_output_folder` (*)

**Note:** meta info param ‘output_extensions’ is by default txt, so that the temporary txt json files get properly deleted

```python
_execute()
```
Downloads files via http

```python
ursgal.resources.platform_independent.arc_independent.get_http_files_1_0_0.get_http_files_1_0_0.
```

### Meta Engines

#### Combine FDR 0_1

```python
class ursgal.wrappers.combine_FDR_0_1.combined_FDR_0_1(*args, **kwargs)
```

**combined 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”.

```python
_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

```python
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

```python
class ursgal.wrappers.combine_pep_1_0_0.combined_pep_1_0_0(*args, **kwargs)
```

**combined_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.

### Misc Engines

#### Filter CSV 1_0_0

```python
class ursgal.wrappers.filter_csv_1_0_0.filter_csv_1_0_0(*args, **kwargs)
filter_csv_1_0_0 UNode
Filters .csv files row-wise according to user-defined rules.

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

_execute()
Result files (.csv) are filtered for defined filter parameters.

_input file has to be a .csv

Creates a _accepted.csv file and returns its path. If defined also rejected entries are written to _rejected.csv.
```
Note: To write the rejected entries define `write_unfiltered_results` as True in the parameters.

Available rules:
- lte
- gte
- lt
- gt
- contains
- contains_not
- equals
- equals_not
- regex

Example

```python
>>> 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

```python
>>> params = {
>>>     'csv_filter_rules': [
>>>         ['Sequence', 'contains_not', 'T'],
>>>         ['Sequence', 'contains_not', 'Y']
>>>     ]
>>> }
```

lte:
‘lower than or equal’ (\(\leq\)) 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’ (\(\geq\)) 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’.

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 occurrence in the database. This ensures that no unequal distribution of target and decoy peptides is present. Further, every peptide is shuffled, while the amino acids where the enzyme cleaves are maintained 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

Available 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.

Merge CSVS 1_0_0

```python
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
```
Merges ident csvs

Sanitize CSV 1.0.0

class ursgal.wrappers.sanitize_csv_1_0_0.sanitize_csv_1_0_0(*args, **kwargs)
sanitize_csv_1_0_0 UNode

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

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

_execute()

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

Input file has to be a .csv

Creates a _sanitized.csv file and returns its path.

**Note:** If not specified, the validation_score_field and bigger_scores_better parameters are determined from the last engine. Therefore, if sanitize_csv_1_0_0 is applied to merged or processed result files, both parameters need to be specified.

Available parameters:

- **score_diff_threshold (float): minimum score difference between** the best PSM and the first rejected PSM of one spectrum

- **threshold_is_log10 (bool): True, if log10 scale has been used for** score_diff_threshold.

- **accept_conflicting_psms (bool): If True, multiple PSMs for one spectrum** can be reported if their score difference is below the threshold. If False, all PSMs for one spectrum are removed if the score difference between the best and second best 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 engines) for the same spectrum are removed. An identification is defined by the combination of ‘Sequence’, ‘Modifications’ and ‘Charge’.**
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 1_0_0**

```python
class ursgal.wrappers.unify_csv_1_0_0.unify_csv_1_0_0(*args, **kwargs)
    unify_csv_1_0_0 UNode
    Unifies the .csv files of converted search engine results. The corrections for each engine are listed in the node under ursgal/resources/platform_independent/arc_independent/unify_csv_1_0_0
    _execute()
    Result files from search engines are unified to contain the same informations in the same style
    Input file has to be a .csv
    Creates a _unified.csv file and returns its path
```

**Upeptide mapper v1_0_0**

```python
class ursgal.wrappers.upeptide_mapper_1_0_0.upeptide_mapper_1_0_0(*args, **kwargs)
    upeptide_mapper_1_0_0 UNode
```

---

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

---

**Available converter classes of upeptide_mapper_1_0_0**

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

```python
    _execute()
    Peptides from search engine csv file are mapped to the given database(s)
```

5.1. Included engines
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)**

```python
class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_mapper_1_0_0.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/](https://pypi.python.org/pypi/pyahocorasick/) for more information.

```python
cache_database(fasta_database)
Function to cache the given fasta database.

    Parameters fas
data_database *(str)* – path to the fasta database

    Note: If the same fasta_name is buffered again all info is purged from the class.
```

```python
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 of peptides is mapped to the same database (fasta_name).

Examples:

```python
def peptide_2_protein_mappings(PEPTIDE):
    return {
        'start': 1,
        'end': 10,
        'pre': 'K',
        'post': 'D',
        'id': 'BSA'
    }
```

Mapper class V3 (dev)

```python
class 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.

```python
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.
```

```python
cache_database(fasta_database, 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))
```

5.1. Included engines
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 of peptides is mapped to the same database (fasta_name).

Examples:

```python
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)**

```python
class ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.upeptide_mapper_1_0_0.UPeptideMapper_v2(word_len=None)
```

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 deprecated 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.

```python
_create_fcache(id=None, seq=None, fasta_name=None)
```

Updates the fast cache with a given sequence

```python
_format_hit_dict(seq, start, end, id)
```

Creates a formatted 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:

```python
{
    'start': 12,
    'end': 18,
    'id': 'Protein Id passed to the function',
    'pre': 'A',
    'post': 'D',
}
```
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:

```python
{
    'start': 12,
    'end': 18,
    'id': 'Protein Id passed to the function',
    'pre': 'A',
    'post': 'V',
}
```

map_peptides(peptide_list, fasta_name=None, force_regex=False)
Wrapper function to map a given peptide list in one batch.

Parameters

- **peptide_list** (list) – list with peptides to be mapped
- **fasta_name** (str) – name of the database

purge_fasta_info(fasta_name)
Purges regular sequence lookup and fcache for a given fasta_name

Quantification Engines

pyQms 1.0.0

Validation Engines

Kojak tailored Percolator 2.08

```python
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 directly 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.

5.1. Included engines
Note: Percolator (2.08) has to be symlinked or copied to engine-folder ‘kojak_percolator_2_08’ in order to make this node work.


```
postflight()
    Convert the percolator output .tsv into the .csv format with headers as in the unified csv format.
```

```
preflight()
    Formatting the command line to via self.params
```

**Percolator 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)


```
postflight()
    read the output and merge in back to the ident csv
```

```
preflight()
    Formatting the command line to via self.params
```

**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


```
postflight()
    Parse the qvality output and merge it back into the csv file
```

```
preflight()
    Formatting the command line to via self.params
```

**Visualizer**

**Plot pyGCluster heatmap from CSV 1_0_0**

```
class ursgal.wrappers.plot_pygcluster_heatmap_from_csv_1_0_0.plot_pygcluster_heatmap_from_csv_1_0_0(*args, **kwargs)
    plot_pygcluster_heatmap_from_csv_1_0_0 UNode
```

```execute()
```
Venn Diagram v1_0_0

```python
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
```

Create's 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 access
by e.g. dict[‘C-(A|B|D)’][‘results’]

’C-(A|B|D)’

or for 2 or 3 or 5 VennDiagrams the appropriate combinations . . .

5.1. Included engines
5.2 How to extend and create new engines?

5.2.1 Create/Implement your own UNode

Before implementing your own UNode, make sure that you have read about the General structure of Urgan. This page will explain how to integrate a standalone executable or Python script into Urgan'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+ v9797

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. Platform 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 is 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': '[AGIL]|{P}',
            'formic_acid': '[D]|{P}',
            'gluc': '[DE]|{P}',
            'glucbicarb': '[E]|{P}',
            'iodosobenzoate': '[W]|{X}',
            'lysc': '[K]|{P}',
            'lysp': '[K]|{X}',
            'lysn': '[X]|{K}',
            'lysn_promisc': '[X]|{AKRS}',
            'nonspecific': '[X]|{X}',
            'pepsina': '[FL]|{X}',
            'protein_endopeptidase': '[P]|{X}',
            'staph_protease': '[E]|{X}',
    },
}]

(continues on next page)
If a parameter is not yet present in uparams.py, you can add a new parameter containing all necessary information (see here).

Example add write_unfiltered_results for filter_csv_1_0_0

```python
'write_unfiltered_results' : {
    'edit_version' : 1.00,
    'available_in_unode' : [
        'filter_csv_1_0_0',
    ],
    'triggers_rerun' : True,
    'ukey_translation' : {
        'filter_csv_style_1' : 'write_unfiltered_results',
    },
    'utag' : [
        'conversion',
    ],
    'uvalue_translation' : {
        'uvalue_type' : 'bool',
        'uvalue_option' : {
            'default_value' : False,
            'description' : 
                'Writes rejected results if True',
        },
    },
},
```

After changing uparams.py, please run the tests, especially chk_format_node_param_test.py to check for errors.

3. Implementation of the wrapper class

Each UNode has to have a Python wrapper file located in:

• `<ursgal_path>/wrappers/ <unode_name>.py`

The UNode has to inherit from the UNode class, which during initialization injects the node related data into the class.

The default structure of the UNode class has to be:

```python
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
```

(continues on next page)
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:

```python
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',
        '{original_engine_parameter}'.format(**self.params['translations']['_grouped_by_translated_key']),
    ]

After *preflight()* Ursgal automatically passes the command_list to Python’s built-in subprocess module:

```python
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.:

```python
def postflight(self):
    '''
    Move the result files to the Kojak folder, since the output files can 
    not be specified manually.
    '''
    # kojak_extensions = [
    #    '.kojak.txt',
    #    '.pep.xml',
    #    '.perc.inter.txt',
    #    '.perc.intra.txt',
    #    '.perc.loop.txt',
    #    '.perc.single.txt',
    #]
    for extension in self.META_INFO['all_extensions']:
        org_path = os.path.join(
            self.params['input_dir_path'],
            '{0}{1}'.format(
                self.params['file_root'],
                extension
            )
        )
        new_path = os.path.join(
            self.params['output_dir_path'],
            '{0}_kojak_{1}{2}'.format(
                self.params['file_root'],
                self.META_INFO['version'],
                extension
            )
        )
        if os.path.exists(org_path):
            shutil.move(
                org_path,
                new_path
            )
```

Example: ursgal/engines/msgfplus_v9979.py

```python
#!/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
    Reference:
    Kim S, Mischerikow N, Bandeira N, Navarro JD, Wich L, Mohammed S, Heck AJ,
    →Mass Spectra: Applications to Database Search.
    '''
    META_INFO = {
        'edit_version' : 1.00,
        'name' : 'MSGF+',
    }
```

(continues on next page)
'version' : 'v9979',
'release_date' : '2010-12-1',
'engine_type' : {
  'protein_database_search_engine' : True,
},
'input_extensions' : ['.mgf', '.mzML', '.mzXML', '.ms2', '.pkl', '.dta.txt'],
'output_extensions' : ['.mzid'],
'create_own_folder' : True,
'in_development' : False,
'include_in_git' : False,
'utranslation_style' : 'msgfplus_style_1',
'engine' : {
  'platform_independent' : {
    'arc_independent' : {
      'exe' : 'MSGFPlus.jar',
      'url' : 'http://proteomics.ucsd.edu/Software/MSGFPlus/MSGFPlus.zip',
      'zip_md5' : '82a3e2204ff698e260ac9f89d3880b59',
      'additional_exe' : [],
    },
  },
},
'citation' : 
  'Applications to Database Search.',

def __init__(self, *args, **kwargs):
  super(msgfplus_v9979, self).__init__(*args, **kwargs)
  pass

def preflight( self ):
  
  Formatting the command line via self.params
  Modifications file will be created in the output folder
  Returns:
  dict: self.params

  translations = self.params['translations']['_grouped_by_translated_key']

  self.params['command_list'] = [
    'java',
    '-jar',
    self.exe,
  ]

  self.params['translations']['mgf_input_file'] = os.path.join( 
    self.params['input_dir_path'],
    self.params['input_file']
  )
  translations['-s']['mgf_input_file'] = self.params['translations']['mgf_input_ --file']

5.2. How to extend and create new engines ?
self.params['translations']['output_file_incl_path'] = os.path.join(
    self.params['output_dir_path'],
    self.params['output_file']
)
translations['-o']['output_file_incl_path'] = self.params['translations'][
    'output_file_incl_path'
]

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

mods_file = open( self.params['translations']['modification_file'], 'w',
    encoding = 'UTF-8' )
modifications = []

if self.params['translations']['label'] == '15N':
    for aminoacid, N15_Diff in ursgal.ukb.DICT_15N_DIFF.items():
        existing = False
        for mod in self.params['mods'][ 'fix' ]:
            if aminoacid == mod[ 'aa' ]:
                mod[ 'mass' ] += N15_Diff
                mod[ 'name' ] += '_15N_' + format(aminoacid)
                existing = True
            if existing == True:
                continue
        else:
            modifications.append( '{0},{1},fix,any,15N_{1}'.format(N15_Diff,
                aminoacid) )

    for t in [ 'fix', 'opt' ]:
        for mod in self.params[ 'mods' ][ t ]:
            modifications.append( '{0},{1},{2},{3},{4}'.format(mod[ 'mass' ],
                mod[ 'aa' ], t, mod[ 'pos' ], mod[ 'name' ] )
)

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

translations['-t'] = {
    '-t': '{0}(1), {2}(1)'.format(
        translations['-t'][ 'precursor_mass_tolerance_minus' ],
        translations['-t'][ 'precursor_mass_tolerance_unit' ],
        translations['-t'][ 'precursor_mass_tolerance_plus' ],
    )
}

command_dict = { (continues on next page)
for translated_key, translation_dict in translations.items():
    if translated_key == '-Xmx':
        self.params[ 'command_list' ].insert(1,'{0}|{1}'.format(translation_key,
            list(translation_dict.values())[0])
    elif translated_key in ['label', 'NumMods']:
        continue
    elif len(translation_dict) == 1:
        command_dict[translated_key] = str(list(translation_dict.values())[0])
    else:
        print('The translated key ', translated_key, ' maps on more than one 
˓
→ukey, but no special rules have been defined')
        print(translation_dict)
        exit(1)
for k, v in command_dict.items():
    self.params[ 'command_list' ].extend((k, v))
return self.params

3.2 Implementation of a UNode from Python code

Using sys.argv or the argparse module, any Python code can be executed like a command line tool. Thus, it is possible
to include pure Python UNodes using the steps described above. For convenience, it is also possible to import
the main function of a Python script using self.import_engine_as_python_function(). This function can then be directly
executed by Ursgal, which makes it possible to include Python scripts that don’t use argparse or sys.argv. To skip
command line execution and run the main function of a Python script, one has to define the _execute() method of the
UNode class. There are several pure Python UNodes in Ursgal, e.g. filter_csv_1_0_0.py, get_ftp_files_1_0_0.py and
many others.

Example: ursgal/engines/filter_csv_1_0_0.py

#!/usr/bin/env 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
Examples

Ursgal comes with multiple example scripts which can be used to test its functionality. Example scripts can also be used as templates for your own scripts.

7.1 Example Scripts

7.1.1 Simple Example Scripts

Simple example search

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

import ursgal
import os
import sys
import shutil

def main():
   
   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)

   ```
input_file

usage:
   ./simple_example_search.py

Note:
   Myrimatch does not work with this file.
   To use MSAmada 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
            '*,,prot-N-term,Acetyl',  # N-Acetylation
        ],
        # '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
    )

(continues on next page)
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).
'''

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': 'https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_˓
    →with_contaminants_target_decoy.fasta',
    'http_output_folder': os.path.join(
        os.pardir,
        'example_data'
    )
}

if os.path.exists(params['ftp_output_folder']) is False:
    os.mkdir(params['ftp_output_folder'])

uc = ursgal.UController(
    profile='LTQ XL low res',
    params=params
)

mzML_file = os.path.join(
    params['ftp_output_folder'],
    params['ftp_include_ext'][0]
)

if os.path.exists(mzML_file) is False:
    uc.fetch_file(
        engine='get_ftp_files_1_0_0'
    )

if os.path.exists(params['database']) is False:
    uc.fetch_file(
        engine='get_http_files_1_0_0'
    )

(continues on next page)
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('	Combined results can be found here:')
print(combined_results)
return

if __name__ == '__main__':
    main()

Target decoy generation

target_decoy_generation_example.main()

Simple example script how to generate a target decoy database.

Note: By default a ‘shuffled peptide preserving cleavage sites’ database is generated. For this script a ‘reverse protein’ database is generated.

usage:

./target_decoy_generation_example.py

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

import ursgal
import os

def main():
    '''
    Simple example script how to generate a target decoy database.

    Note: By default a 'shuffled peptide preserving cleavage sites' database is
    generated.
    '''

7.1. Example Scripts 93
A generated. For this script a 'reverse protein' database is generated.

usage:

./target_decoy_generation_example.py

params = {
    'enzyme': 'trypsin',
    'decoy_generation_mode': 'reverse_protein',
}

fasta_database_list = [
    os.path.join(  
        os.pardir,  
        'example_data',  
        'BSA.fasta'  
    )
]

uc = ursgal.UController(
    params=params
)

new_target_decoy_db_name = uc.execute_misc_engine(
    input_file=fasta_database_list,
    engine='generate_target_decoy_1_0_0',
    output_file_name='my_BSA_target_decoy.fasta',
)

print('Generated target decoy database: {0}'.format(new_target_decoy_db_name))

if __name__ == '__main__':
    main()
 Ursgal Documentation, Release ursgal_version = '0.6.2'

(continued from previous page)

```python
'M, opt, any, Oxidation', # Met oxidation
'C, fix, any, Carbamidomethyl', # Carbamidomethylation
'*', opt, Prot-N-term, Acetyl' # N-Acetylation
],
},
)

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(
    input_file=mzML_file, # from OpenMS example files
    engine='mzml2mgf_1_0_0'
)
for prefix in ['10ppm', '20ppm']:
    R.params['prefix'] = prefix
    output_file = R.search(
        input_file=mgf_file,
        engine=engine,
        # output_file_name = 'some_userdefined_name'
    )

# Second method: Automatically convert to MGF inside the loop:
# (MGF conversion is re-run every time because the prefix changed!)
for prefix in ['5ppm', '15ppm']:
    R.params['prefix'] = prefix
    output_file = R.search(
        input_file=mzML_file, # from OpenMS example files
        engine=engine,
    )
```

(continues on next page)
Test node execution

test_node_execution.main()

Testscript for executing the test node, which also tests the run time determination function.

Usage: ./test_node_execution.py

7.1.2 Complete Workflow Scripts

Do it all folder wide

do_it_all_folder_wide.main(folder=None, profile=None, target_decoy_database=None)

An example test script to search all mzML files which are present in the specified folder. The search is currently performed on 4 search engines and 2 validation engines.

The machine profile has to be specified as well as the target-decoy database.

usage: 

./do_it_all_folder_wide.py <mzML_folder> <profile> <target_decoy_database>

Current profiles:

• ‘QExactive+’
• ‘LTQ XL low res’
• ‘LTQ XL high res’
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
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
)

# 1. complete workflow:
# 2. every spectrum file is searched with every search engine,
# 3. results are validated (for each engine separately),
# 4. 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.execute_misc_engine(
            input_file=validated_results,
            engine='merge_csvs_1_0_0',
        )
        filtered_validated_results = uc.execute_misc_engine(
            input_file=validated_results_from_all_engines,
            engine='filter_csv_1_0_0',
        )
        result_files.append(filtered_validated_results)

results_all_files = uc.execute_misc_engine(
    input_file=result_files,
    engine='merge_csvs_1_0_0',
)

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

`barth_et_al_large_scale.main(folder)`

Example script for reproducing the data for figure 3

usage:

```
./barth_et_al_large_scale.py <folder>
```

The folder determines the target folder where the files will be downloaded

Chlamydomonas reinhardtii samples

Three biological replicates of 4 conditions (2_3, 2_4, 3_1, 4_1)


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

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

import ursgal
import glob
import os.path
import sys

def main(folder):
    '''
    Example script for reproducing the data for figure 3
    usage:
    
    ./barth_et_al_large_scale.py <folder>
    
    The folder determines the target folder where the files will be downloaded
    Chlamydomonas reinhardtii samples
    
    (continues on next page)
```
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 peptidatlas

```
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.peptidatlas.org',
    'ftp_login': 'PASS00269',
    'ftp_password': 'FI4645a',
    'ftp_output_folder_root': folder,
    'http_url': 'https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_
    →with_contaminants_target_decoy.fasta',
    'http_output_folder': os.path.join(
        os.pardir,
        'example_data'
    ),
}
```

uc = ursgal.UController(
    params=input_params
)
if os.path.exists(input_params['database']) is False:
    uc.fetch_file(
        engine='get_http_files_1_0_0'
    )

output_folder_to_file_list = {
    ('rep1_sample_2_3', 'LTQ XL high res'): [
        'CF_07062012_ph8_2_3A.mzML',
        'CF_13062012_ph3_2_3A.mzML',
        'CF_13062012_ph4_2_3A.mzML',
        'CF_13062012_ph5_2_3A.mzML',
        'CF_13062012_ph6_2_3A.mzML',
        'CF_13062012_ph11FT_2_3A.mzML',
    ],
    ('rep1_sample_2_4', 'LTQ XL high res'): [
        'CF_07062012_ph8_2_4A.mzML',
        'CF_13062012_ph3_2_4A_120615113039.mzML',
        'CF_13062012_ph4_2_4A.mzML',
        'CF_13062012_ph5_2_4A.mzML',
        'CF_13062012_ph6_2_4A.mzML',
        'CF_13062012_ph11FT_2_4A.mzML',
    ],
    ('rep1_sample_3_1', 'LTQ XL high res'): [
        'CF_12062012_ph8_1_3A.mzML',
        'CF_13062012_ph3_1_3A.mzML',
        'CF_13062012_ph4_1_3A.mzML',
        'CF_13062012_ph5_1_3A.mzML',
        'CF_13062012_ph6_1_3A.mzML',
        'CF_13062012_ph11FT_1_3A.mzML',
    ],
    ('rep1_sample_4_1', 'LTQ XL high res'): [
        'CF_07062012_ph8_1_4A.mzML',
        'CF_13062012_ph3_1_4A.mzML',
        'CF_13062012_ph4_1_4A.mzML',
        'CF_13062012_ph5_1_4A.mzML',
        'CF_13062012_ph6_1_4A.mzML',
        '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, profile)}

...
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': [],
    'myrimatch_2_1_138': [],
    'msgfplus_v9979': [],
    'msamanda_1_0_0_5243': [],
}

two_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.execute_misc_engine(
        input_file=unified_results_list,
        engine="merge_csvs_1_0_0",
    )

    # Validation with Percolator:
    percolator_validated = uc.validate(
        input_file=merged_unified,
        engine='percolator_2_08',  # one could replace this with 'qvality'
    )
    percolator_results[search_engine].append(
        percolator_validated
    )

    # At this point, the analysis is finished. We got
    # Percolator-validated results for each of the 3
    # replicates and 12 conditions.

    # But let's see how well the five search engines
    # performed! To compare, we collect all PSMs with
    # an estimated FDR <= 0.01 for each engine, and
    # plot this information with the VennDiagram UNode.
    # We will also use the Combine FDR Score method
    # to combine the results from all five engines,
    # and increase the number of identified peptides.

    five_large_merged = []
    filtered_final_results = []

    # We will estimate the FDR for all 60 files
    # (5 engines x12 files) when using percolator PEPs as
    # quality score
    uc.params['validation_score_field'] = 'PEP'
    uc.params['bigger_scores_better'] = False

    # To make obtain smaller CSV files (and make plotting
    # less RAM-intensive, we remove all decoys and PSMs above
    # 0.06 FDR
    uc.params['csv_filter_rules'] = [
        ['estimated_FDR', 'lte', 0.06],
        ['Is decoy', 'equals', 'false']
    ]
    for engine, percolator_validated_list in percolator_results.items():
        # unfiltered files for cFDR script
twelve_merged = uc.execute_misc_engine(
    input_file=percolator_validated_list,
    engine='merge_csvs_1_0_0',
)

twelve_filtered = []
for one_of_12 in percolator_validated_list:
    one_of_12_FDR = uc.validate(
        input_file=one_of_12,
        engine='add_estimated_fdr_1_0_0'
    )
    one_of_12_FDR_filtered = uc.execute_misc_engine(
        input_file=one_of_12_FDR,
        engine='filter_csv_1_0_0'
    )
    twelve_filtered.append(one_of_12_FDR_filtered)

# For the combined FDR scoring, we merge all 12 files:
filtered_merged = uc.execute_misc_engine(
    input_file=twelve_filtered,
    engine='merge_csvs_1_0_0'
)

five_large_merged.append(twelve_merged)
filtered_final_results.append(filtered_merged)

# The five big merged files of each engine are combined:
cFDR = uc.combine_search_results(
    input_files=five_large_merged,
    engine='combine_FDR_0_1',
)

# We estimate the FDR of this combined approach:
uc.params['validation_score_field'] = 'Combined FDR Score'
uc.params['bigger_scores_better'] = False

cFDR_FDR = uc.validate(
    input_file=cFDR,
    engine='add_estimated_fdr_1_0_0'
)

# Removing decoys and low quality hits, to obtain a
# smaller file:
uc.params['csv_filter_rules'] = [
    ['estimated_FDR', 'lte', 0.06],
    ['Is decoy', 'equals', 'false']
]
cFDR_filtered_results = uc.execute_misc_engine(
    input_file=cFDR_FDR,
    engine='filter_csv_1_0_0',
)
filtered_final_results.append(cFDR_filtered_results)

# Since we produced quite a lot of files, let’s print the full
# paths to our most important result files so we find them quickly:
print(
    ''',
)
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.

```python
print("\n
########### Result files: ####################")
for result_file in filtered_final_results:
    print('\t*{0}'.format(result_file))
```

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

---

**Complete workflow for human BR dataset analysis**

human_br_complete_workflow.main(folder)

**usage:**

```
./human_br_complete_workflow.py <folder_with_human_br_files>
```

This script produces the data for figure 3.

```python
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import os
import sys
import pprint

def main(folder):
    
    usage:

    ./human_br_complete_workflow.py <folder_with_human_br_files>

    This script 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)
        sys.exit(1)

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

# The target peptide database which will be searched (UniProt Human
# reference proteome from July 2013)
target_database = 'uniprot_human_UP000005640_created_until_20130707.fasta'
# Let's turn it into a target decoy database by reversing peptides:
target_decoy_database = uc.execute_misc_engine(
    input_file=target_database,
    engine='generate_target_decoy_1_0_0')

# OMSSA parameters from Wen et al. (2015):
omssa_params = {
    # (used by default) # -w
    'he': '1000',   # -he 1000
    'zcc': '1',     # -zcc 1
    'frag_mass_tolerance': '0.6',  # -to 0.6
    'frag_mass_tolerance_unit': 'da',  # -to 0.6
    'precursor_mass_tolerance_minus': '10',  # -te 10
    'precursor_mass_tolerance_plus': '10',  # -te 10
    'precursor_mass_tolerance_unit': 'ppm',  # -teppm
    'score_a_ions': False,  # -i 1,4
    'score_b_ions': True,   # -i 1,4
    'score_c_ions': False,  # -i 1,4
    'score_x_ions': False,  # -i 1,4
    'score_y_ions': True,   # -i 1,4
    'score_z_ions': False,  # -i 1,4
    'enzyme': 'trypsin_p',  # -e 10
    'maximum_missed_cleavages': '1',  # -v 1
    'precursor_max_charge': '8',  # -zh 8
}


# MS-GF+ parameters from Wen et al. (2015):
msgf_params = {
    'precursor_min_charge': '1',  # -zi 1
    'tez': '1',  # -tez 1
    'precursor_isotope_range': '0,1',  # -ti 1
    'num_match_spec': '1',  # -hc 1
    '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',  # fragment ion mass tolerance was set to 0.6 Da
    'frag_mass_tolerance_unit': 'da',  # fragment ion mass tolerance was set to 0.6 Da
    'precursor_max_charge': '8',  # maximum parent charge of spectrum was set as 8
    'enzyme': 'trypsin',  # the enzyme was set as trypsin
    'database': target_decoy_database,
    'modifications': modifications,
}

(continues on next page)
'maximum_missed_cleavages': '1',
# (used by default) # no model refinement was employed.

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

search_engine_settings = [
# not used in Wen et al., so we use the same settings as xtandem
('msamanda_1_0_0_5243', xtandem_params, 'LTQ XL high res'),
# not used in Wen et al., so we use the same settings as xtandem
('myrimatch_2_1_138', xtandem_params, 'LTQ XL high res'),
# the instrument selected was High-res
('msgfplus_v9979', msgf_params, 'LTQ XL high res'),
('xtandem_jackhammer', xtandem_params, None),
('omssa_2_1_9', omssa_params, None),
]

merged_validated_files_3_engines = []
merged_validated_files_5_engines = []

for engine, wen_params, instrument in search_engine_settings:
    # Initializing the uPLANIT UController class with
    # our specified modifications and mass spectrometer
    uc = ursgal.UController(
        params=wen_params
    )
    if instrument is not None:
        uc.set_profile(instrument)
    unified_results = []
    percolator_validated_results = []
    for mzML_file in mass_spec_files:
        unified_search_results = uc.search(
            input_file=mzML_file,
            engine=engine,
        )
        unified_results.append(
            unified_search_results
        )
        validated_csv = uc.validate(
            input_file=unified_search_results,
            engine='percolator_2_08',
        )
        percolator_validated_results.append(validated_csv)
    merged_validated_csv = uc.execute_misc_engine(
        input_file=percolator_validated_results,
        engine='merge_csvs_1_0_0'
    )
    merged_unvalidated_csv = uc.execute_misc_engine(
        input_file=unified_results,
        engine='merge_csvs_1_0_0',
    )

(continues on next page)
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__)
        sys.exit(1)
    main(sys.argv[1])

Example search for 15N labeling and no label

search_with_label_15N.main()

Executes a search with 3 different search engines on an example file from the data from Barth et al. Two searches are performed per engine for each 14N (unlabeled) and 15N labeling. The overlap of identified peptides for the 14N and 15N searches between the engines is visualized as well as the overlap between all 14N and 15N identified peptides.

usage: ./search_with_label_15N.py

Note: It is important to convert the mgf file outside of (and before :) ) the search loop to avoid mgf file redundancy and to assure correct retention time mapping in the unify_csv node.

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

import ursgal
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 per 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:
 Note: It is important to convert the mgf file outside of (and before :) ) the search loop to avoid mgf file redundancy and to assure correct retention time mapping in the unify_csv node.

```
engine_list = [  
    'omssa_2_1_9',  
    'xtandem_piledriver',  
    'msgfplus_v9979',  
]

params = {  
    'database': os.path.join(  
        os.pardir,  
        'example_data',  
        'Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta'  
    ),  
    'modifications': [],  
    'csv_filter_rules': [  
        ['PEP', 'lte', 0.01],  
        ['Is decoy', 'equals', 'false']  
    ],  
    'ftp_url': 'ftp.peptideatlas.org',  
    'ftp_login': 'PASS00269',  
    'ftp_password': 'FI4645a',  
    'ftp_include_ext': [  
        'JB_FASP_pH8_2-3_28122012.mzML',  
    ],  
    'ftp_output_folder': os.path.join(  
        os.pardir,  
        'example_data',  
        'search_with_label_15N'  
    ),  
    'http_url': 'https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_  
            with_contaminants_target_decoy.fasta',  
    'http_output_folder': os.path.join(  
        os.pardir,  
        'example_data'  
    )  
}

if os.path.exists(params['ftp_output_folder']) is False:  
    os.mkdir(params['ftp_output_folder'])

uc = ursgal.UController(  
    profile='LTQ XL low res',  
    params=params
)

mzML_file = os.path.join(  
    params['ftp_output_folder'],  
    params['ftp_include_ext'][0]
)

if os.path.exists(mzML_file) is False:  
    # (continues on next page)  
```
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(
    input_file=mzML_file,
    engine='mzml2mgf_1_0_0',
)
files_2_merge = {}
label_list = ['14N', '15N']
for label in label_list:
    validated_and_filtered_files_list = []
    uc.params['label'] = label
    uc.params['prefix'] = label
    for engine in engine_list:
        search_result = uc.search_mgf(
            input_file=mgf_file,
            engine=engine,
        )
        converted_result = uc.convert(
            input_file=search_result,
            guess_engine=True,
        )
        mapped_results = uc.execute_misc_engine(
            input_file=converted_result,
            engine='upeptide_mapper',
        )
        unified_search_results = uc.execute_misc_engine(
            input_file=mapped_results,
            engine='unify_csv',
        )
        validated_file = uc.validate(
            input_file=unified_search_results,
            engine='percolator_2_08',
        )
        filtered_file = uc.execute_misc_engine(
            input_file=validated_file,
            engine='filter_csv',
        )
        validated_and_filtered_files_list.append(filtered_file)
    files_2_merge[label] = validated_and_filtered_files_list
uc.visualize(
    input_files=validated_and_filtered_files_list,
    engine='venndiagram',
)
uc.params['prefix'] = None
uc.params['label'] = ''
uc.params['visualization_label_positions'] = {}
label_comparison_file_list = []
for n, label in enumerate(label_list):
    uc.params['visualization_label_positions'][str(n)] = label
    label_comparison_file_list.append(
        uc.execute_misc_engine(  
            engine='perseus'  
        )
    )
uc.params['visualization_label_positions'] = 
uc.params['visualization_label_positions'].append(label_comparison_file_list)
uc.visualize(
    input_files=label_comparison_file_list,
    engine='venndiagram',
)

(continues on next page)
7.1.3 Version Comparison Scripts

X!Tandem version comparison

```
xtrandem_version_comparison.main()
Executes a search with 5 versions of X!Tandem on an example file from the
data from Barth et al. 2014.

usage: ./xtandem_version_comparison.py

This is a simple example file to show the straightforward comparison of different program versions of X!Tandem
Creates a Venn diagram with the peptides obtained by the different versions.

Note: At the moment in total 6 XTandem versions are incorporated in Ursgal.
```

```python
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import os

def main():
    '''
    Executes a search with 5 versions of X!Tandem on an example file from the
data from Barth et al. 2014.

    usage: ./xtandem_version_comparison.py

    This is a simple example file to show the straightforward comparison of different program versions of X!Tandem
    Creates a Venn diagram with the peptides obtained by the different versions.

    Note: At the moment in total 6 XTandem versions are incorporated in Ursgal.
    '''
```
engine_list = [
    '# xtandem_cyclone',
    'xtandem_jackhammer',
    'xtandem_sledgehammer',
    'xtandem_piledriver',
    'xtandem_vengeance',
    'xtandem_alanine',
]

params = {
    'database': os.path.join(
        os.pardir,
        'example_data',
        'Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta'
    ),
    'modifications': [],
    'csv_filter_rules': [
        ['PEP', 'lte', 0.01],
        ['Is decoy', 'equals', 'false']
    ],
    'ftp_url': 'ftp.peptideatlas.org',
    'ftp_login': 'PASS00269',
    'ftp_password': 'FI4645a',
    'ftp_include_ext': [
        'JB_FASP_pH8_2-3_28122012.mzML',
    ],
    'ftp_output_folder': os.path.join(
        os.pardir,
        'example_data',
        'xtandem_version_comparison'
    ),
    'http_url': 'https://www.sas.upenn.edu/~sschulze/Creinhardtii_281_v5_5_CP_MT_→with_contaminants_target_decoy.fasta',
    'http_output_folder': os.path.join(
        os.pardir,
        'example_data'
    )
}

if os.path.exists(params['ftp_output_folder']) is False:
    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(continued on next page)
filtered_files_list = []
for engine in engine_list:
    unified_result_file = uc.search(
        input_file=mzML_file,
        engine=engine,
    )
    validated_file = uc.validate(
        input_file=unified_result_file,
        engine='percolator_2_08',
    )
    filtered_file = uc.execute_misc_engine(
        input_file=validated_file,
        engine='filter_csv_1_0_0',
    )
    filtered_files_list.append(filtered_file)

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

if __name__ == '__main__':
    main()
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 high resolution data
which can be better handled by version newer than Jackhammer. One gains
approximately 10 percent more peptides with newer versions of X!Tandem.

    Creates a Venn diagram with the peptides obtained by the different versions.
    '''

    required_example_file = 'B_D140314_SGSDSsample1_R01_MSG_T0.mzML'

    if os.path.exists(os.path.join(folder, required_example_file)) is False:
        print('Your specified folder does not contain the required example file: {}
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])

7.1.4 Parameter Optimization Scripts

BSA machine ppm offset example

bsa_ppm_offset_test.main()
Example script to do a simple machine ppm offset parameter sweep. The m/z values in the example mgf file are stepwise changed and the in the final output the total peptides are counted.

usage: ./bsa_ppm_offset_test.py

Note: As expected, if the offset becomes to big no peptides can be found anymore.
(-9, '-9_ppm_offset'),
(-8, '-8_ppm_offset'),
(-7, '-7_ppm_offset'),
(-6, '-6_ppm_offset'),
(-5, '-5_ppm_offset'),
(-4, '-4_ppm_offset'),
(-3, '-3_ppm_offset'),
(-2, '-2_ppm_offset'),
(-1, '-1_ppm_offset'),
(None, '0_ppm_offset'),
(1, '1_ppm_offset'),
(2, '2_ppm_offset'),
(3, '3_ppm_offset'),
(4, '4_ppm_offset'),
(5, '5_ppm_offset'),
(6, '6_ppm_offset'),
(7, '7_ppm_offset'),
(8, '8_ppm_offset'),
(9, '9_ppm_offset'),
(10, '10_ppm_offset'),
]

engine_list = [
    'xtandem_vengeance'
]

R = ursgal.UController(
    profile='LTQ XL low res',
    params={
        'database': os.path.join(
            os.pardir,
            'example_data',
            'BSA.fasta')
    } ,
    modifications: [
        'M,opt,any,Oxidation',   # Met oxidation
        'C,fix,any,Carbamidomethyl', # Carbamidomethylation
        '*',opt,Prot-N-term,Acetyl'  # N-Acetylation
    ]
)

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
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()
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-Acetylitaion
            ],
        }
    )
    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'

(continues on next page)
R.params['http_output_folder'] = os.path.dirname(mzML_file)
R.fetch_file(
    engine='get_http_files_1_0_0'
)
try:
    shutil.move(
        '{0}?format=raw'.format(mzML_file),
        mzML_file
    )
except:
    shutil.move(
        '{0}?format=raw'.format(mzML_file),
        mzML_file
)

# Convert mzML to MGF outside the loop, so this step is not repeated in
# the loop
mgf_file = R.convert(
    input_file=mzML_file,
    engine='mzml2mgf_1_0_0'
)

for engine in engine_list:
    for precursor_mass_tolerance in precursor_mass_tolerance_list:
        R.params['precursor_mass_tolerance_minus'] = precursor_mass_tolerance
        R.params['precursor_mass_tolerance_plus'] = precursor_mass_tolerance
        R.params['prefix'] = '{0}_precursor_mass_tolerance_'.format(precursor_mass_tolerance)
        unified_search_result_file = R.search(
            input_file=mgf_file,
            engine=engine,
            force=False,
        )

        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('
'.join(['Search with {0: >2} ppm precursor mass tolerance found {1: >2} peptides'.format(tolerance, len(peptide_set)) if ident in peptide_set else '' for ident in 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 too small, very few peptides are found. With this small sweep the actual
min accuracy of a mass spectrometer can be estimated.

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

import ursgal
import glob
import csv
from collections import defaultdict as ddict
import os
import shutil

def main():
    '''
    Example script to do a fragment mass tolerance parameter sweep.
    
    usage:
    ./bsa_fragment_mass_tolerance_example.py
    
    If the fragment mass tolerance becomes too small, very 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': [  
```
(continues on next page)
{'M', 'opt', 'any', 'Oxidation', 'C', 'fix', 'any', 'Carbamidomethyl', '*', 'opt', 'Prot-N-term', 'Acetyl'},
]
}
)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/...
    R.params['http_output_folder'] = os.path.dirname(mzML_file)
    R.fetch_file(
        engine='get_http_files_1_0_0'
    )
try:
    shutil.move(
        '{0}?format=raw'.format(mzML_file),
        mzML_file
    )
except:
    shutil.move(
        '{0}format=raw'.format(mzML_file),
        mzML_file
    )

# Convert mzML to MGF outside the loop, so this step is not repeated in
# the loop
mgf_file = R.convert(
    input_file=mzML_file,
    engine='mzml2mgf_1_0_0',
)
for engine in engine_list:
    for fragment_mass_tolerance in fragment_mass_tolerance_list:
        R.params['frag_mass_tolerance'] = fragment_mass_tolerance
        R.params['prefix'] = '{0}_fragment_mass_tolerance_'.format(
            fragment_mass_tolerance
        )
        unified_search_result_file = R.search(
            input_file=mgf_file,
            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')):
Bruderer et al. (2015) machine offset sweep (data for figure 2)

machine_offset_bruderer_sweep.search(input_folder=None)

Does the parameter sweep on every tenth MS2 spectrum of the data from Bruderer et al. (2015) und X!Tandem Sledgehammer.

Note: Please download the .RAW data for the DDA dataset from peptideatlas.org (PASS00589, password: WF6554orn) and convert to mzML. Then the script can be executed with the folder with the mzML files as the first argument.

Warning: This script (if the sweep ranges are not changed) will perform 10080 searches which will produce approximately 100 GB output (inclusive mzML files)

usage:

./machine_offset_bruderer_sweep.py <folder_with_bruderer_data>

Sweeps over:

- machine_offset_in_ppm from -20 to +20 ppm offset
- precursor mass tolerance from 1 to 5 ppm
- fragment mass tolerance from -2.5 to 20 ppm

The search can be very time consuming (depending on your machine/cluster), therefor the analyze step can be performed separately by calling analyze() instead of search() when one has already performed the searches and wants to analyze the results.

machine_offset_bruderer_sweep.analyze(folder)

 Parses the result files form search and write a result .csv file which contains the data to plot figure 2.
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'),
    (8.33, '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.AGFufeza/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
    ""

(continues on next page)
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), therefore 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.

```python
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 MQ_offset, file_name in MQ_OFFSET_TO_FILENAME:
        file_2_target_offset[file_name]['offsets'].append(n / 1e6)
```
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(
        input_file=mzML_path,
        engine='mzml2mgf_1_0_0'
    )

for engine in engine_list:
    for precursor_ion_tolerane in precursor_ion_tolerance_list:
        for frag_ion_tolerance in frag_ion_tolerance_list:

            new_prefix = prefix_format_string.format(
                precursor_ion_tolerane,
                frag_ion_tolerance
            )

            R.params["precursor_mass_tolerance_minus"] = precursor_ion_tolerane
            R.params["precursor_mass_tolerance_plus"] = precursor_ion_tolerane
            R.params["frag_mass_tolerance"] = frag_ion_tolerance
            R.params["prefix"] = new_prefix

            unified_search_result_file = R.search(
                input_file=mzML_path,
                engine=engine,
                force=False,
            )

return

def analyze(folder):
    '''
    Parses the result files form search and write a result .csv file which contains the data to plot figure 2.
    '''

    R = ursgal.UController(
        profile='QExactive+',
        params=GENERAL_PARAMS
    )
    csv_collector = {}
    ve = 'quality_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(''.join((line_dict['Sequence'], line_dict['Modifications'])))

fieldnames = [}
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__)
        sys.exit(1)
    search(sys.argv[1])
    analyze(sys.argv[1])

7.1.5 Filter CSV Examples

Filter for modifications

filter_csv_for_mods_example.main()
Examples script for filtering unified results for modification containing entries

usage: ./filter_csv_for_mods_example.py
Will produce a file with only entries which contain Carbamidomethyl as a modification.
#!/usr/bin/env python3
# encoding: utf-8

import ursgal
import os

def main():
    '''
    Examples script for filtering unified results for modification containing entries
    
    usage:
    ./filter_csv_for_mods_example.py
    
    Will produce a file with only entries which contain Carbamidomethyl as a modification.
    '''
    params = {
        'csv_filter_rules': [
            ['Modifications', 'contains', 'Carbamidomethyl'],
            'write_unfiltered_results': False
        ]
    }

csv_file_to_filter = os.path.join(os.pardir, 'example_data', 'misc', 'filter_csv_for_mods_example_omssa_2_1_9_pmap_unified.csv')
uc = ursgal.UController(params=params)
filtered_csv = uc.execute_misc_engine(input_file=csv_file_to_filter, engine='filter_csv_1_0_0',)

if __name__ == '__main__':
    main()

Filter validated results

filter_csv_validation_example.main()
    Examples script for filtering validated results for a PEP <= 0.01 and remove all decoys.
    
    usage: ./filter_csv_validation_example.py
    
    Will produce a file with only target sequences with a posterior error probability of lower or equal to 1 percent

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

(continues on next page)
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.execute_misc_engine(input_file=csv_file_to_filter, engine='filter_csv.1.0.0')

if __name__ == '__main__':
    main()

7.1.6 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)

Simple 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)

```python
#!/usr/bin/env python3
# encoding: utf-8
import ursgal
import csv
from collections import defaultdict as ddict
import os
import glob
import math
params = {
    'database': os.path.join(
        os.pardir,
        'example_data',
        'Creinhardtii_281_v5_5_CP_MT_with_contaminants_target_decoy.fasta',
    ),
    'csv_filter_rules': [
        ['Is decoy', 'equals', 'false'],
        ['PEP', 'lte', 0.01],
    ],
}
# We specify all search engines and validation engines that we want to use in a list (version numbers might differ on windows or mac):
search_engines = [
    'omssa',
    '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',
    ]
```
'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'
],
}

def get_files():
    uc = ursgal.UController(params=get_params)
    if os.path.exists(params['database']) is False:
        uc.fetch_file(
            engine='get_http_files_1_0_0'
        )
    if os.path.exists(get_params['ftp_output_folder']) is False:
        os.makedirs(get_params['ftp_output_folder'])
        uc.fetch_file(
            engine='get_ftp_files_1_0_0'
        )
    spec_files = []

(continues on next page)
for mzML_file in glob.glob(
    os.path.join(
        get_params['ftp_output_folder'],
        '*.mzML'
    )
):
    spec_files.append(mzML_file)

return spec_files

def search(validation_engine):
    '''
    Executes a cascade search on four example files from the
data from Barth et al.

    usage:
    ./cascade_search_example.py

    Searches for peptides using a cascade search approach similar to Kertesz-Farkas et al.
    for which spectra were first searched for unmodified peptides, followed by
    consecutive searches
    for the following modifications:
    oxidation of M,
    deamidation of N/Q,
    methylation of E/K/R,
    N-terminal acetylation,
    phosphorylation of S/T.
    After each step, spectra with a PSM below 1 % PEP were removed.
    '''
    # Initializing the uPLANIT UController class with
    # our specified modifications and mass spectrometer
    uc = ursgal.UController(
        profile=mass_spectrometer,
        params=params
    )
    # complete workflow for every level of the cascade:
    # every spectrum file is searched with every search engine,
    # results are validated seperately,
    # validated results are merged and filtered for targets and PEP <= 0.01.

def workflow(spec_file, prefix=None, validation_engine=None, filter_before_validation=False, force=False):
    validated_results = []

    # Convert mzML to MGF outside the loop, so this step is not repeated in
    # the loop
    mgf_spec_file = uc.convert(
        input_file=spec_file,
        engine='mzml2mgf_1_0_0'
    )

    for search_engine in search_engines:
        uc.params['prefix'] = prefix
        unified_search_results = uc.search(
            input_file=mgf_spec_file,
            engine=search_engine,
        )

        # (continues on next page)
    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.execute_misc_engine(
        input_file=unified_search_results,
        engine='filter_csv_1_0_0'
    )
else:
    filtered_search_results = unified_search_results

validated_search_results = uc.validate(
    input_file=filtered_search_results,
    engine=validation_engine,
    force=force,
)
validated_results.append(validated_search_results)

validated_results_from_all_engines = uc.execute_misc_engine(
    input_file=sorted(validated_results),
    engine='merge_csvs_1_0_0',
    force=force,
)

uc.params['csv_filter_rules'] = [
    ['Is decoy', 'equals', 'false'],
    ['PEP', 'lte', 0.01],
]

filtered_validated_results = uc.execute_misc_engine(
    input_file=validated_results_from_all_engines,
    engine='filter_csv_1_0_0'
)
return filtered_validated_results

result_files = []
for spec_file in spec_files:
    spectra_with_PSM = set()
    for level in sorted(cascade.keys()):
        uc.params['modifications'] = cascade[level]
        if level == '0':
            results = workflow(
                spec_file,
                validation_engine=validation_engine,
                prefix='cascade-lvl-{0}'.format(level)
            )
        else:
            uc.params['scan_exclusion_list'] = list(spectra_with_PSM)
            results = workflow(
                spec_file,
                validation_engine=validation_engine,
                filter_before_validation=True,
                force=True,
                prefix='cascade-lvl-{0}'.format(level)
            )
result_files.append(results)
# spectrum IDs for PSMs are written into an exclusion list for the next level of the cascade search,
# these spectra will be 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 cascade level: {}
'.format(len(spectra_with_PSM)))

results_all_files = uc.execute_misc_engine(
    input_file=sorted(result_files),
    engine='merge_csvs_1_0_0',
)
return results_all_files

def analyze(collector):
    
    Simle analysis script for the cascade search,
    counting the number of identified peptides (combination of peptide sequence and modifications)
    and PSMs (additionally include the spectrum ID)
    
    mod_list = ['Oxidation', 'Deamidated', 'Methyl', 'Acetyl', 'Phospho']
    fieldnames = ['approach', 'count_type', 'validation_engine', 'unmodified', 'multimodified'] + mod_list + ['total']

    csv_writer = csv.DictWriter(open('cascade_results.csv', 'w'), fieldnames)
    csv_writer.writeheader()
    uc = ursgal.UController()
    uc.params['validation_score_field'] = 'PEP'
    uc.params['bigger_scores_better'] = False

    # Count the number of identified peptides and PSMs for the different modifications
    # Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score is counted
    # and only if the best hit has a PEP that is at least two orders of magnitude smaller than the others
    for validation_engine, result_file in collector.items():
        
(continues on next page)
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:
            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']
    )
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)
os.pardir,
'example_data',
'C. reinhardtii_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': 'https://www.sas.upenn.edu/~sschulze/C. reinhardtii_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(params['ftp_output_folder']) is False:
        os.makedirs(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(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.execute_misc_engine(
    input_file=sorted(validated_results),
    engine='merge_csvs',
)
filtered_validated_results = uc.execute_misc_engine(
    input_file=validated_results_from_all_engines,
    engine='filter_csv'
)
result_files.append(filtered_validated_results)

results_all_files = uc.execute_misc_engine(
    input_file=sorted(result_files),
    engine='merge_csvs',
)
return results_all_files

def analyze(collector):
    '''
    Simple analysis script for the ungrouped search,
    counting the number of identified peptides (combination of peptide sequence and modifications)
    and PSMs (additionally include the spectrum ID)
    '''
    mod_list = ['Oxidation', 'Deamidated', 'Methyl', 'Acetyl', 'Phospho']
    fieldnames = ['approach', 'count_type', 'validation_engine', 'unmodified', 'multimodified'] + mod_list + ['total']

csv_writer = csv.DictWriter(open('ungrouped_results.csv', 'w'), fieldnames)
csv_writer.writeheader()
uc = ursgal.UController()
uc.params['validation_score_field'] = 'PEP'
uc.params['bigger_scores_better'] = False

# Count the number of identified peptides and PSMs for the different modifications
# Spectra with multiple PSMs are sanitized, i.e. only the PSM with best PEP score is counted
# and only if the best hit has a PEP that is at least two orders of magnitude smaller than the others
for validation_engine, result_file in collector.items():
    counter_dict = {
        'psm': ddict(set),
        'pep': ddict(set)
    }
grouped_psms = uc._group_psms(
    result_file,
validation_score_field='PEP',
bigger_scores_better=False
)

for spec_title, grouped_psm_list in grouped_psms.items():
    best_score, best_line_dict = grouped_psm_list[0]
    if len(grouped_psm_list) > 1:
        second_best_score, second_best_line_dict = grouped_psm_list[1]
        best_peptide_and_mod = best_line_dict['Sequence'] + best_line_dict['Modifications']
        second_best_peptide_and_mod = second_best_line_dict['Sequence'] + second_best_line_dict['Modifications']

        if best_peptide_and_mod == second_best_peptide_and_mod:
            line_dict = best_line_dict
        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:
            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 = {
(continues on next page)
'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 separately.

grouped_search_example.analyze(collector)
Simple 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)
```
},
    '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'),
    }
```

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-{}'.format(group)
            filtered_results = uc.execute_misc_engine(
                input_file=unified_search_results,
                engine='filter_csv'
            )
            uc.params['prefix'] = ''
            validated_search_results = uc.validate(
                input_file=filtered_results,
                engine=validation_engine,
            )
            validated_results.append(validated_search_results)
        uc.params['prefix'] = 'file{}'.format(n)
        validated_results_from_all_engines = uc.execute_misc_engine(0, spec_file, engine=search_engine,)
(continues on next page)
def analyze(collector):
    '''
    Simple 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 subset of data from Barth et al. 2014.

<table>
<thead>
<tr>
<th>approach</th>
<th>count_type</th>
<th>validation_engine</th>
<th>un-modified</th>
<th>multi-modified</th>
<th>Oxidation</th>
<th>Deamidated</th>
<th>Methyl</th>
<th>Acetyl</th>
<th>Phospho</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ungrouped</td>
<td>psm</td>
<td>qvality</td>
<td>6905</td>
<td>24</td>
<td>270</td>
<td>157</td>
<td>42</td>
<td>23</td>
<td>6</td>
<td>7427</td>
</tr>
<tr>
<td>grouped</td>
<td>psm</td>
<td>qvality</td>
<td>8009</td>
<td>267</td>
<td>128</td>
<td>31</td>
<td>27</td>
<td>3</td>
<td>8465</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>psm</td>
<td>qvality</td>
<td>8009</td>
<td>316</td>
<td>131</td>
<td>46</td>
<td>30</td>
<td>1</td>
<td>7788</td>
<td></td>
</tr>
<tr>
<td>ungrouped</td>
<td>upep</td>
<td>qvality</td>
<td>1409</td>
<td>14</td>
<td>94</td>
<td>48</td>
<td>20</td>
<td>11</td>
<td>3</td>
<td>1599</td>
</tr>
<tr>
<td>grouped</td>
<td>upep</td>
<td>qvality</td>
<td>1575</td>
<td>93</td>
<td>42</td>
<td>9</td>
<td>17</td>
<td>1</td>
<td>1737</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>upep</td>
<td>qvality</td>
<td>1552</td>
<td>102</td>
<td>44</td>
<td>13</td>
<td>13</td>
<td>1</td>
<td>1725</td>
<td></td>
</tr>
</tbody>
</table>

Human BR dataset results.

<table>
<thead>
<tr>
<th>approach</th>
<th>count_type</th>
<th>validation_engine</th>
<th>un-modified</th>
<th>multi-modified</th>
<th>Oxidation</th>
<th>Deamidated</th>
<th>Methyl</th>
<th>Acetyl</th>
<th>Phospho</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ungrouped</td>
<td>psm</td>
<td>qvality</td>
<td>18813</td>
<td>51</td>
<td>332</td>
<td>336</td>
<td>102</td>
<td>292</td>
<td>122</td>
<td>20048</td>
</tr>
<tr>
<td>grouped</td>
<td>psm</td>
<td>qvality</td>
<td>21370</td>
<td>383</td>
<td>171</td>
<td>100</td>
<td>435</td>
<td>142</td>
<td>22601</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>psm</td>
<td>qvality</td>
<td>19065</td>
<td>424</td>
<td>251</td>
<td>111</td>
<td>417</td>
<td>38</td>
<td>20406</td>
<td></td>
</tr>
<tr>
<td>ungrouped</td>
<td>upep</td>
<td>qvality</td>
<td>6707</td>
<td>39</td>
<td>143</td>
<td>173</td>
<td>27</td>
<td>123</td>
<td>58</td>
<td>7270</td>
</tr>
<tr>
<td>grouped</td>
<td>upep</td>
<td>qvality</td>
<td>7525</td>
<td>161</td>
<td>104</td>
<td>31</td>
<td>191</td>
<td>58</td>
<td>8070</td>
<td></td>
</tr>
<tr>
<td>cascade</td>
<td>upep</td>
<td>qvality</td>
<td>7784</td>
<td>172</td>
<td>152</td>
<td>43</td>
<td>171</td>
<td>55</td>
<td>8377</td>
<td></td>
</tr>
</tbody>
</table>
7.1.7 Upeptide Mapper Example Scripts

Upeptide mapper v3 and 4 complete C. reinhardtii proteome match

`complete_chlamydomonas_proteome_match.main(class_version)`
Example script to demonstrate speed and memory efficiency of the new upeptide_mapper.

All tryptic peptides (n=1,094,395, 6 < len(peptide) < 40 ) are mapped to the Chlamydomonas reinhardtii (38876 entries) target-decoy database.

```
usage: ./complete_chlamydomonas_proteome_match.py <class_version>
```

Class versions

- UPeptideMapper_v2
- UPeptideMapper_v3
- UPeptideMapper_v4

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
Record of released Ursgal versions with all notable changes

8.1 Changelog

8.1.1 Version 0.6.2 (09.2018)

1. MSFragger version 20171106 was implemented
2. MODa v1.61 was implemented
3. PIPI version 1.4.5 replaced PIPI 1.3 due to compatibility issues with PIPI 1.3
4. PIPI version 1.4.6 implemented (thanks Fengchao)
5. Added latest versions of MSGF+ (version 2018.06.28 and 2018.09.12) and new python mzid to csv converter msgfplus2csv_py_v1_0_0
6. Macot wrapper and converter was implemented to allow downstream processing of Mascot results
7. Some minor bugfixes and added functionality (check git log for details)

8.1.2 Version 0.6.1 (03.2018)

1. Improved compatibility to pymzML generation 2 and splitted up the mzML to mgf converter in version 1.0.0 (pymzML 0.7.9) and 2.0.0 (pymzML 2.0.2). The version of installed pymzML is now automatically determined and the corresponding converter version is used.
2. Added latest versions of MSGF+ (version 2018.01.20) and the corresponding C based mzidentML converter (version 1.2.0 and 1.2.1)
3. Several new engine versions were made available via the install_resources script.
4. Some general code cleanup was done
8.1.3 Version 0.6.0 (01.2018)

1. Restructuring of engine classes. SEARCH_ENGINE(s) are now divided into * CROSS_LINK_SEARCH_ENGINE(s) * DE_NOVO_SEARCH_ENGINE(s) * PROTEIN_DATABASE_SEARCH_ENGINE(s) * SPECTRAL_LIBRARY_SEARCH_ENGINE(s) The META_INFO of corresponding engines has been changed accordingly. Furthermore, CONVERTER(s) have been split into CONVERTER(s) and MISC_ENGINE(s)

2. Restructuring of UController functions. Unified functions for all engines of one engine class (Converter, Search_Engines, Validation_Engines, etc) are now available. Function names for each engine class are stored in ukb.ENGINE_TYPES and ukb.UCONTROLLER_FUNCTIONS

3. ursgal_kb.py has been renamed to ukb.py

4. Implemented the following open modification search engines (as protein database search engines): MSFragger, PIPI, ModA

5. Smaller fixes and improvements (please check git log)

8.1.4 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 node. Classes for mapping can be imported from anywhere from the undoe. Input for standalone node is a not-unified csv file. Branch: upapa_v3

3. Unify csv is now placed after the upeptide_mapper node if a database search engine (e.g. OMSSA, X!Tandem etc.) is used. Branch: upapa_v3

4. Unify csv was adjusted to meet the new requirements of the separated peptide mapping node. Please also note, that the default behaviour of remapping amino acid ‘U’ to ‘C’ is not longer performed.

5. Unify csv now reports if the peptide fulfills the enzyme cleavage parameters, like number of missed cleavages and if the C and N terminus is correct. Column name: ‘Complies search criteria’

6. Test script update

7. Documentation update

8. Implementation of a customizable SVM for PSM post-processing

9. Smaller fixes and improvements (please check git log)

8.1.5 Version 0.4.0rc1 (05.2016)

1. included a upeptide_mapper for fast peptide to sequence mapping.

2. renamed engine folder to wrappers

3. combined all files from the kb folder into one single file, uparams.py which is parsed during unode initialization. Advantage is to see all params grouped together.

4. Added more information to the unique parameters, such as description and default value types.

5. Included script to auto-generate documentation from uparams file.

6. Updated documentation to reflect the changes above.
8.1.6 Version 0.3.4 (02.2016)

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

9.1 Contribution Guidelines

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

9.1.1 Summary

In general, contribution to Ursgal is very welcome! Feel free to fork and/or clone Ursgal. If you want to improve code or contribute new nodes/tools/algorithms please read these guidelines first. If something is unclear please contact one of the authors for help or let us know via e.g. an issue.

We are happy to include your name to the list of contributors in the README. Drop a line to one of the developers if you want to get included (and of course you actually contributed something).

9.1.2 Commit messages

First of all, please be concise and as descriptive (explicit is better than implicit :) ) as possible. It is always helpful to point out, which parts of Ursgal were changed/fixed (e.g. documentation or example scripts etc. ). In the same time, please avoid unnecessarily long messages.

9.1.3 Parameters

The central idea of Ursgal are the unified parameters. The central parameter is translated, so that every engine can use it. This means, if you implement a new engine, you have to go through the (more or less) tedious process to check, if parameter X of the new engine Y is already listed in uparams.py. We require to be very thorough in this process. Having the same parameter multiple times must be avoided! There may be difficult cases, to decide if the parameter is actually the same, but by using the translation system in Ursgal, some adjustments can be made. Please refer to the documentation for further instructions and considerations on the parameters.
9.1.4 Code standards and conventions

Since this a collaborative project, you will encounter different coding styles. Despite the fact that we know that diversity is beautiful, we need to keep some common line on how to code (This list may be further extended). We generally use PEP8 style (https://www.python.org/dev/peps/pep-0008/) with the exception of E203 (whitespaces before : in order to align values in dicts). Additionally this list will give you some things to think about:

- Re-think naming of variables at least twice
- Re-check deleting of own debug code before sending Pull requests
- Re-check own files created by nosetests and add it into `.gitignore` before sending Pull requests

9.1.5 Test philosophy

Test your code! Seriously, test you code! If you add new functionality or nodes at the same time provide (a) test function(s). We have already a set of tests and different files, which can be used for the test. Avoid adding new test files if possible to keep the repo small.

9.1.6 Sphinx guide

We use Sphinx to automatically build and format the documentation. Please keep this style in your docstrings

9.1.7 Other rules and cosiderations

None so far.

9.1.8 Merge/pull requests

Please use the pull request to push your code to the master repository. It will be automatically tested by Travis and AppVeyor if the module is still working in unix and Windows environments. Pull requests will be discussed by the main dev team and merged into Ursgal.

9.1.9 Issues

If you have an issue or problem, please first search all open issues and pull request to avoid duplication of efforts. If you have a fix for the problem you may directly open a pull request. On the other hand, if you plan to or are already working on implementing new stuff, you may also open an issue and (pre-) announce your contribution. Please tag then the issue with ‘enhancement’. In general the core team of Ursgal will also take care of crucial bugs in the main code. Since Ursgal is open source, we cannot maintain every detail and assure its compatibility and functionality (please be reminded here to test your code, seriously, test your code)

9.1.10 Citation

Be reminded, that in an academic world, citations are the only credit that one can hope for ;) Therefore, please make sure to properly cite every tool that you use or implement. And of course, if you use Ursol, do not forget to cite us

10.1 Frequently Asked Questions

10.1.1 Installation

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

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

Q: MS-GF+ (or any Java based engine) fails. What could be the problem?

To run Java based engines like MS-GF+ or MSFragger, Java Runtime Environment needs to be installed. Visit http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html for download and installation.

Q: Downloading http files is not working on OSX. Why?

Make sure that certificates are properly installed. Go to Applications/Python 3.6 and double-click Install Certificates.command. The latest version of Python3.6 for Mac should come with the right certifications for secure connections anyways.

Q: I have problems installing pyahocorasick on Windows! What can I do?

Generally, Python 3.6 should be used when working with Windows. Here are some general remarks for a flawless installation under Windows.


When using Windows 10, consider additionally installing MS Build Tools 2015
10.1.2 Usage

Q: Found mismatch between json parameter . . . .

| Found mismatch between json parameter csv_filter_rules:          |
| [(['PEP', 'lte', 0.01], ['Is decoy', 'equals', 'false'])] and |
| controller params csv_filter_rules:                             |
| [('PEP', 'lte', 0.01), ('Is decoy', 'equals', 'false')].       |
| Consider re-run with force=True or delete old u.jsons.          |

During JSON dump Python tuples are converted into list like objects, thus this might be a reason. Just change your parameter to lists instead of tuples :)

Q: How do I add an engine that is not installed via install_resources.py?

Download the engine from the respective developers homepage (links are given in the engine wrapper documentation). Create a folder in the corresponding Ursgal resources (name of the folder = name of the engine in Ursgal, for more information see Create/Implement your own UNode: 1. Integration into Resources) and unpack/save all required files, especially the executable, there. Remember to run:

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

to include Ursgal (and the changes you have made to the resources) into Python site-packages.

Q: The example script simple_example_search.py fails. What am I doing wrong?

Check the printouts: at which step is it failing? If the download of the example BSA1.mzML was not successful, and you’re using OSX, see Q: Downloading http files is not working on OSX. Why?. If MS-GF+ fails and you are not sure if you have installed Java Runtime Environment, see Q: MS-GF+ (or any Java based engine) fails. What could be the problem?. If this doesn’t help, shoot us a message or open an issue on GitHub (please include your printouts).

Q: A validation engine (Percolator, qvality, . . . ) fails. What’s going on?

There are two common problems causing your workflow to fail at the point of validating results: 1. Your database doesn’t contain decoys (check out target_decoy_generation_example.py) or decoys are not recognized (check if the uparam ‘decoy_tag’ is correct for your database). 2. Your list of results is too small for proper statistics (the error message is something like “Too good seperation between targets and decoys”). In this case, you need to improve your search parameters (e.g. mass tolerances), database size (e.g. whole proteome instead of a single protein) or MS measurements (i.e. your raw data).

10.1.3 Development

Q: How do I create/add a new engine?

See Create/Implement your own UNode.

Q: How do I keep Ursgal up-to-date?

Ursgal is still in development and changes, extensions, etc. are pushed to GitHub. Therefore, the easiest way (if you have cloned Ursgal from GitHub) is:
user@localhost:~/ursgal$ git pull

If you have not cloned Ursgal but used the ZIP file you can replace the folder with the newly downloaded and extracted version.

In both cases you might need to run the setup again to update the python site-packages:

user@localhost:~/ursgal$ python3 setup.py install
11.1 Known Issues

11.1.1 General

- Java used memory size
  
  Adjust the memory usage by Java according to your needs. When using memory intensive tasks as mzIdentML conversion of large files, an adjustment 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):


11.1.2 Windows general

- Some modules can not be compiled/installed with python3.4+ using pip.
  
  E.g. pyahocorasick can not be installed. This has the consequence, that on Windows the old peptide mapper version is used. There are several workarounds to compile and install the modules manually, e.g.:

  http://haypo-notes.readthedocs.io/python.html#build-a-python-wheel-package-on-windows

11.1.3 Windows 10

- MS Amanda can not load .fasta files
• **calculating the md5 can cause problems e.g. while executing test.** This is due to different line endings on Unix and Windows systems. The test functions test for both md5, so this problem should be avoided.

### 11.1.4 MONO

• **Mono is the .NET replacement under *nix systems, since .NET is not directly** ported by Microsoft to other systems than windows. Unfortunately mono is not as stable as the official .NET build. Therefore:

  MS Amanda crashes randomly under *nix systems (e.g. Linux or OS X)
u
ursgal.ucore, 42
Symbols

__init__() (ursgal.UNode method), 36
__weakref__ (ursgal.UNode attribute), 36
_create_fcache() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 74
_execute() (ursgal.UNode method), 36
_execute() (ursgal.wrappers.combine_FDR_0_1.combine_FDR_0_1 method), 65
_execute() (ursgal.wrappers.combine_pep_1_0_0.combine_pep_1_0_0 method), 66
_execute() (ursgal.wrappers.csv2counted_results_1_0_0.csv2counted_results_1_0_0 method), 59
_execute() (ursgal.wrappers.csv2ssl_1_0_0.csv2ssl_1_0_0 method), 59
_execute() (ursgal.wrappers.filter_csv_1_0_0.filter_csv_1_0_0 method), 66
_execute() (ursgal.wrappers.generate_target_decoy_1_0_0.generate_target_decoy_1_0_0 method), 68
_execute() (ursgal.wrappers.get_ftp_files_1_0_0.get_ftp_files_1_0_0 method), 64
_execute() (ursgal.wrappers.get_http_files_1_0_0.get_http_files_1_0_0 method), 65
_execute() (ursgal.wrappers.merge_csvs_1_0_0.merge_csvs_1_0_0 method), 69
_execute() (ursgal.wrappers.plot_pygcluster_heatmap_from_csv_1_0_0.plot_pygcluster_heatmap_from_csv_1_0_0 method), 76
_execute() (ursgal.wrappers.unify_csv_1_0_0.unify_csv_1_0_0 method), 63, 71
_execute() (ursgal.wrappers.unpeptide_mapper_1_0_0.unpeptide_mapper_1_0_0 method), 71
_execute() (ursgal.wrappers.venndiagram_1_0_0.venndiagram_1_0_0 method), 77
_format_hit_dict() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 74
_group_psms() (ursgal.UNode method), 36
A
abs_paths_for_specific_keys() (ursgal.UNode method), 37
add_glycan() (ursgal.ChemicalComposition method), 45
appMass2id_list() (ursgal.UnimodMapper method), 47
appMass2name_list() (ursgal.UnimodMapper method), 47
B
build_lookup() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 75
build_lookup_from_file() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 75
C
cache_database() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 73
cache_database() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 72
calc_md5() (ursgal.UNode method), 37
calculate_mz() (in module ursgal.ucore), 42
Collect_and_translate_params() (ursgal.UNode method), 45
collect_and_translate_params() (ursgal.UNode method), 45
cache_database() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 73
cache_database() (ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0.UPeptideMapper_v2 method), 72
calc_md5() (ursgal.UNode method), 37
calculate_mz() (in module ursgal.ucore), 42
ChemicalComposition (class in ursgal), 44
clear() (ursgal.ChemicalComposition method), 45
collection_and_translate_params() (ursgal.UNode method), 45
collection_and_translate_params() (ursgal.UNode method), 45
combine_search_results() (ursgal.ucontroller.UController method), 23
compare_json_and_local_ursgal_version() (ursgal.UNode method), 37
composition2id_list() (ursgal.UnimodMapper method), 47
composition2mass() (ursgal.UnimodMapper method), 47
composition2name_list() (ursgal.UnimodMapper method), 48
composition_at_pos (ursgal.ChemicalComposition attribute), 45
composition_of_aa_at_pos (ursgal.ChemicalComposition attribute), 45
composition_of_mod_at_pos (ursgal.ChemicalComposition attribute), 46
convert() (ursgal.ucontroller.UController method), 24
convert_dalton_to_ppm() (in module ursgal.ucore), 42
convert_ppm_to_dalton() (in module ursgal.ucore), 42
convert_results_to_csv() (ursgal.ucontroller.UController method), 24
distinguish_multi_and_single_input() (ursgal.ucontroller.UController method), 25
determine_availability_of_unodes() (ursgal.ucontroller.UController method), 25
determine_common_name() (ursgal.UNode method), 37
determine_common_top_level_folder() (ursgal.UNode method), 37
digest() (in module ursgal.ucore), 42
distinguish_multi_and_single_input() (ursgal.ucontroller.UController method), 25
download_resources() (ursgal.ucontroller.UController method), 25
dump_json_and_calc_md5() (ursgal.UNode method), 38
dump_multi_json() (ursgal.ucontroller.UController method), 25
eengine_sanity_check() (ursgal.ucontroller.UController method), 25
eval_if_run_needs_to_be_executed() (ursgal.ucontroller.UController method), 26
execute_mis_engine() (ursgal.ucontroller.UController method), 26
execute_unode() (ursgal.ucontroller.UController method), 26
F
fetch_file() (ursgal.ucontroller.UController method), 27
filter_csv() (ursgal.ucontroller.UController method), 27
filter_csv_1_0_0 (class in ursgal.wrappers.filter_csv_1_0_0), 66
flatten_list() (ursgal.UNode method), 38
format_templates() (ursgal.wrappers.kojak_1_5_3.kojak_1_5_3 method), 59
format_templates() (ursgal.wrappers.xtandem_piledriver.xtandem_piledriver method), 57
format_templates() (ursgal.wrappers.xtandem_sledgehammer.xtandem_sledgehammer method), 57
format_templates() (ursgal.wrappers.xtandem_vengeance.xtandem_vengeance method), 57
G
generate_multi_file_dicts() (ursgal.ucontroller.UController method), 28
generate_multi_helper_file() (ursgal.ucontroller.UController method), 28
generate_target_decoy() (ursgal.ucontroller.UController method), 28
generate_target_decoy_1_0_0 (class in ursgal.wrappers.generate_target_decoy_1_0_0), 68
generate_target_decoy_1_0_0 (class in ursgal.wrappers.generate_target_decoy_1_0_0), 68
get_ftp_files_1_0_0 (class in ursgal.wrappers.get_ftp_files_1_0_0), 64
get_http_files_1_0_0 (class in ursgal.wrappers.get_http_files_1_0_0), 64
get_last_engine() (ursgal.UNode method), 38
get_last_engine_type() (ursgal.UNode method), 38
get_last_search_engine() (ursgal.UNode method), 39
get_masked_params() (ursgal.umapmaster.UParamMapper method), 43
get_mzml_that_corresponds_to_mgf() (ursgal.ucontroller.UController method), 29
group_styles() (ursgal.umapmaster.UParamMapper method), 43
guess_engine_name() (ursgal.ucontroller.UController method), 29
H
hill_notation() (ursgal.ChemicalComposition method), 46
hill_notation_unimod() (ursgal.ChemicalComposition method), 46
id2composition() (ursgal.UnimodMapper method), 48
unify_csv() (ursgal.ucontroller.UController method), 34
unify_csv_1_0_0 (class in ursgal.wrappers.unify_csv_1_0_0), 63, 71
UnimodMapper (class in ursgal), 46
UNode (class in ursgal), 36
UParamMapper (class in ursgal.umapmaster), 43
update_params_with_io_data() (ursgal.UNode method), 41
update_output_json() (ursgal.UNode method), 41
upeptide_mapper_1_0_0 (class in ursgal.wrappers.upeptide_mapper_1_0_0), 71
UPeptideMapper_v2 (class in ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0., 74
UPeptideMapper_v3 (class in ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0., 73
UPeptideMapper_v4 (class in ursgal.resources.platform_independent.arc_independent.upeptide_mapper_1_0_0., 72
ursgal.ucore (module), 42
use() (ursgal.ChemicalComposition method), 46
V
validate() (ursgal.ucontroller.UController method), 35
venndiagram_1_0_0 (class in ursgal.wrappers.venndiagram_1_0_0), 77
verify_engine_produced_an_output_file() (ursgal.ucontroller.UController method), 35
visualize() (ursgal.ucontroller.UController method), 35
W
write_param_file() (ursgal.wrappers.myrimatch_2_1_138.myrimatch_2_1_138 method), 55
writeXML() (ursgal.UnimodMapper method), 49
X
xtandem2csv_1_0_0 (class in ursgal.wrappers.xtandem2csv_1_0_0), 62
xtandem_alanine (class in ursgal.wrappers.xtandem_alanine), 56
xtandem_cyclone_2010 (class in ursgal.wrappers.xtandem_cyclone_2010), 58
xtandem_jackhammer (class in ursgal.wrappers.xtandem_jackhammer), 58
xtandem_piledriver (class in ursgal.wrappers.xtandem_piledriver), 57
xtandem_sledgehammer (class in ursgal.wrappers.xtandem_sledgehammer), 57
xtandem_vengeance (class in ursgal.wrappers.xtandem_vengeance), 56