BrEPSdb Documentation

Release 2.0.0

Christian-Alexander Dudek
Department of Bioinformatics & Biochemistry
TU Braunschweig

Feb 10, 2017

CONTENTS

CHAPTER

ONE

LICENSE

CHAPTER

TWO

BREPSDB

2.1 brepsdb.py

This is the main file to call for BrEPS database creation.

For desktop mode just call `python breps.py` in the command line, for the computer cluster (SGE) mode should be deployed via `qsub` from cmaster01 using `sge_starter.bash breps.py`. For example: `qsub sge_starter.bash breps.py`

No further functions or classes are provided.

2.2 data.py

This module handles the download and preparation of all needed data.

The `data` folder is associated to this module.

class `BrEPSdb.data.`**advanced_parser**(*strFile*, *dctRef*)

Parser for the advanced data preparation in the BrEPS 2.0 protocol.

This parser inherits basic functionality from *basic_parser* (page 3) and is intended to be used on TrEMBL flatfiles. The parser needs the dictionray `dctRef` containing TrEMBL accessions based on sequence similarity from a UniRef file.

Requirements:

- Sequence length 100-6999 aa (see `reLength`).
- Cross-reference (RX) entry.

Exclusion:

- Accession is not similar to a Swiss-Prot sequence based on UniRef data in `dctRef`.
- Sequence length does not matches original length +- 25%

Entries are saved with the corresponding Swiss-Prot accession in the source field.

class `BrEPSdb.data.`**basic_parser**(*strFile*)

Basic parser for UniProt files.

The basic parser saves accession and sequence of every UniProt entry with a specific length defined in `reLength`. This is just a proof on concept for more advanced parsers. Methods `_require()`, `_exclude()` and `_process()` are supposed to be overwritten in advanced parsers.

lstFields

list – Two character line identifiers to save while parsing the file.

_exclude(*dctEntry*)

Checks if a data entry needs to be excluded.

Parameters **dctEntry** (*dict*) – A dictionary containing all gathered data fileds.

Returns `True` if the entry needs to be excluded, `False` otherwise.

Return type boolean

_process(*dctEntry*)
Processes the data entry using helper functions.

Parameters **dctEntry** (*dict*) – A dictionary containing all gathered data fileds.

Returns A dictionary with processed data fields.

Return type dict

_require(*dctEntry*)
Checks if all requirements are met for a data entry.

Parameters **dctEntry** (*dict*) – A dictionary containing all gathered data fileds.

Returns `True` if the conditions are met, `False` otherwise.

Return type boolean

parse(*openwith=<class 'gzip.GzipFile'>*)
Starts the parsing of the given UniProt file.

The parser works in three steps for every entry in the UniProt Flatfile.

1. Check if requirements are met.
2. Check if the data needs to be excluded.
3. Process the data.

Successfully prepared datasets are directly saved to database.

Parameters **openwith** (*function*) – Function to open the file. Needs to work just like Pythons `open()` function. (Default: `gzip.GzipFile`)

save()
Saves the parsed data saved in `self.lstData` to database.

Data is saved to the `sequences` table of the working database. The filed names are automatically fetched from the dictionary keys.

BrEPSdb.data.**download**(*strServer*, *strFolder*, *strFile*)
Retrieves a file either from ftp or a local path.

Parameters

- **strServer** (*str*) – FTP Server to connect to, `None` for local file.
- **strFolder** (*str*) – Path to the file, either FTP or local.
- **strFile** (*str*) – File to download from FTP sever or to copy from.

Returns Path to the downloaded file.

Return type str

Raises `IOError` – If could not connect to ftp or folder does not exist.

BrEPSdb.data.**get_release**(*strFile*)
Extracts the release date from strFile.

The file has to be in Uniprots reldate.txt format containing three lines. First line is the release identifier (<year>_<month>, eg. 2015_12) of UniprotKB. The next two lines are the release idetifier for Swiss-Prot and TrEMBL data with corresponding release date (<day>-<month>-<year>, eg. 09-Dec-2015).

Parameters **strFile** (*str*) – Path to the file containing the reldate data.

Returns A tuple containing date of the update in seconds and version identifier.

Return type tuple

BrEPSdb.data.**new_version**()
Downloads current version of UniProt from the FTP server.

Connects to the uniprot server via FTP to retrieve the reldate.txt file containing the release informations.

Returns True if there is a newer release available, False if not.

Return type boolean

class BrEPSdb.data.**original_parser**(*strFile*)
Parser as used by the original BrEPS protocol.

This parser inherits basic functionality from *basic_parser* (page 3).

Requirements:

•Sequence length 100-6999 aa (see reLength).

Exclusion:

•Keywords in DE field: putative, hypothetical, fragment, probable, possible, potential.

Entries are saved as enzymes (filed enzyme=1) if DE field contains a valid EC number.

BrEPSdb.data.**parse_uniref**(*strFile*, *openwith=<class 'gzip.GzipFile'>*)
Very simple parser for UniRef XML files.

The parser function basically fetches accessions in the representativeMember tag and saves corresponding accessions in the member tag.

Todo

Replace the function with a proper XML parser!

Parameters

- **strFile** (*str*) – Path to the UniRef XML file to parse.
- **openwith** (*function*) – Function to open the file. Needs to work just like Pythons open() function. (Default: gzip.GzipFile)

Returns Dictionary containing Trembl accession as key and corresponding SwissProt accession.

Return type dict

Raises IOError – UniRef file doesn't exist.

BrEPSdb.data.**process_acc**(*strAcc*)
Parser helper function. Returns the primary accession from a string.

More infos about primary accessions here: (see http://web.expasy.org/docs/userman.html#AC_line)

Parameters **strAcc** (*str*) – One or more comma separated accessions.

Returns First (primary) accession.

Return type str

BrEPSdb.data.**process_date**(*strDate*)
Parser helper function. Converts dd-mmm-yyyy to timestamp from a string.

Parameters **strDate** (*str*) – Date in dd-mm-yyyy format.

Returns datetime object for the corresponding date.

Return type datetime

BrEPSdb.data.**process_ecs**(*strEcs*)
Parser helper function. Returns all found ec numbers as a string separated by a semicolon

BrEPSdb.data.**process_seq**(*strSeq*)
Parser helper function. Replaces whitespaces from sequence string.

class BrEPSdb.data.**sprot_parser**(*strFile*)
Parser for the advanced data preparation in the BrEPS 2.0 protocol.

This parser inherits basic functionality from *original_parser* (page 5) and is intended to be used on Swiss-Prot flatfiles.

Requirements:

•Sequence length 100-6999 aa (see reLength).

•Protein existence on protein level (PE 1).

Exclusion:

•Keywords in DE field: putative, hypothetical, fragment, probable, possible, potential.

Entries are saved as enzymes (filed enzyme=1) if DE field contains a valid EC number and PE = 1. Non enzymes are saved independent from protein existence.

2.3 blast.py

This module prepares FASTA files and executes blastp for the BLAST alignment.

The blast folder is associated to this module.

BrEPSdb.blast.**blast**(*lstFiles*, *useSGE=False*)
Executes the BLAST either on the SGE or on multiple threads.

All sequences will be written to a database file in FASTA format. In parallel the sequences will be written into <intFiles> separate files for BLASTing on multiple SGE cluster nodes or threads. The database will get masked by the BLAST segmasker to mask repetetive regions in sequences. Finally the database file will be created. SGE jobs are created using sge_blast.bash.

Parameters

- **lstFiles** (*list*) – List of all FASTA files for the BLAST.
- **useSGE** (*boolean*) – Whether or not to deploy BLAST on the SGE.

Returns list of created BLAST result files.

Return type list

BrEPSdb.blast.**prepare**(*intFiles=1*)
Prepares all files for the blast alignment.

All sequences will be written to a database file in FASTA format. In parallel the sequences will be written into <intFiles> separate files for BLASTing on multiple SGE cluster nodes or threads. The database will get masked by the BLAST segmasker to mask repetetive regions in sequences. Finally the database file will be created.

Parameters **intFiles** (*int*) – Number of different sequence files to generate for BLASTing (Default 1).

Returns list of created FASTA files.

Return type list

BrEPSdb.blast.**save**(*lstFiles=None*)
Reads the alignment results and saves the BLAST alignments to the database.

Parses the <lstFiles> and saves alignments to the database. Results for seq1/seq2 and seq2/seq1 will be merged into one result to produce a symmetrical matrix and saving just one side of the matrix with seq1 = min(seq1,seq2), seq2 = max(seq1,seq2) and using the lower E-value. E-values of 0.0 will be 1 for self alignments and 1e-181 for all other alignments.

Parameters **lstFiles** (*list*) – List of files with BLAST results.

Returns True if successfull.

Return type boolean

Raises IOError – If one file of <lstFiles> is not found.

2.4 clustering.py

This module clusters the sequences using the BLAST aligbnment data. The module implements the optimally efficient CLINK algorithm using the MySQL database as data backend. The result is a database assisted CLINK algorithm, specialized for the usage of BLAST E-values as distance measurement.

The data folder is associated to this module.

class BrEPSdb.clustering.**Node**(*seq_id*, *left*, *right*, *dist*, *node_id*)
Represents a node in a cluster.

The outer node is the created cluster. Every node has a left and right child node. If not the node is a leaf in the tree.

seq_id
int – Sequence id, if leaf node, None otherwise.

left
int – Left child node, or None if node is a leaf.

right
int – Right child node, or None if node is a leaf.

dist
float – Distance between left and right child node (E-value).

depth
int – Maximum num of edges downstream of the node.

size
int – Num of nodes downstream of the node.

get_nodes(*lstNodes=None*)
Returns all nodes as a list.

is_leaf()
Returns True if the node has no child nodes. Otherwise False.

to_list()
Returns a nested list of all leaf seq_ids.

A list represents a couple of nodes, for example: [[[[[0, 1], [2, 3]], [6, 7]], [4, 5]], [8, 9]]

to_tuple(*cluster_id*)
Returns a tuple with values of all attributes.

Needs the cluster_id for proper include into the database!

BrEPSdb.clustering.**agglomerate**(*threshold=0.0001*)
Agglomerates the given data into one or more clusters.

Uses clink clustering as described in http://comjnl.oxfordjournals.org/content/16/1/30.short and http://comjnl.oxfordjournals.org/cgi/content/long/20/4/364

Parameters **threshold** (*float*) – Defines the threshold where the agglomeration. will stop. Resulting in disconnected trees. (Default: 1e-4).

Returns Returns a dictionary with all created clusters.

Return type dict

BrEPSdb.clustering.**save**(*dctClusters*)
Takes clusters created by *agglomerate()* (page 7) and saves them to the working database.

Parameters **dctClusters** (*dict*) – A dictionary filled with *Node* (page 7) objects representing the clusters, created by *agglomerate()* (page 7).

2.5 pattern.py

This module clusters prepares FASTA files for the multiple sequence alignments (MSA). The actual MSA is created using Clustal Omega. Based on the MSA output the patterns are created and additional extended patterns are created.

The pattern folder is associated to this module.

BrEPSdb.pattern.**create_pattern**(*strFilename*, *blnRemove=True*)
Processes the MSA result to create the pattern.

Reads the MSA result files and generates patterns based on the aligned sequences. A pattern needs to have a minimum length of 8 positions from the first conserved or semi-conserved position to the last conserved or semi-conserved position.

The consensus positions are intepreted with * (Asterisk) as conserved position, \: (Colon) as semi-conserved position, blank and . (Colon) as non-conserved position.

Parameters **strFilename** (*str*) – Path to MSA file to process.

Returns Tuple with pattern data or empty tuple if no pattern generated.

Return type tuple

BrEPSdb.pattern.**extend**()
Extends all patterns using similarity sets for semi-conserved amino acid positions.

All patterns are fetched and the pattern positions are extended using Gonnets PAM250 similarity matrix. The extended patterns are saved to the working database.

BrEPSdb.pattern.**generate**(*intThreads=1*, *useSGE=False*)
Coordinates pattern generation using SGE or multithreading.

Parameters

- **intThreads** (*int*) – Number of threads for desktop mode (Default: 1).
- **useSGE** (*boolean*) – True if patterns should be generated on SGE computer cluster, False for desktop mode. (Default False)

BrEPSdb.pattern.**get_pairs**(*strGrp*)
Returns all character combinations of a given string.

Parameters **strGroup** (*str*) – String with characters.

Raises list – A list with all character combinations.

BrEPSdb.pattern.**process**(*lstClusters*)
Calls the multiple sequence alignments for cluster nodes.

Reads the nodes created at clustering from the database and creates a fasta file <cluster_id>_<node_id>.fsa containing the associated sequences. Multiple sequence alignment (MSA) will be perormed by clustalo. Pattern will be generated with MSA output file.

Parameters **lstClusters** (*list*) – List of cluster ids to process.

Raises IOError – If no MSA output was generated.

BrEPSdb.pattern.**similarity_set**(*lstMat*, *lstAA*)
Creates a sets of similar amino acids for pattern extension.

Similarity sets are created for all amino acid pairs p_{ij}, if the score `s` in similarity matrix `lstMat` is s_{ij}>0.5. Any other amino acid `x` is added to the set if s_{ix}> s_{ij} **and** s_{jx}> s_{ij}.

Parameters

- **lstMat** (*list*) – Matrix of amino acid similarities.
- **lstAA** (*list*) – List with mapping of amino acids for `lstMat`

Returns Dictionary with lists of similar amino acids. Keys are the amino acid pairs.

Return type dict

2.6 verification.py

This module contains the verification process for all generate patterns.

`BrEPSdb.verification.`**timeout**(*intSecs*, **args*)
Timeout decorator for the pattern verification.

Complex regular expression search may run for a long time causing the verification to get stuck. The decorator adds a signal that kills a verification that runs longer than `intSecs` seconds.

Parameters **intSecs** (*integer*) – Seconds after which the timeout is fired.

`BrEPSdb.verification.`**verificator**(*strTable='patterns'*, *intThreads=1*, *useSGE=False*)
Coordinates pattern verification using SGE or multithreading.

Parameters

- **strTable** (*string*) – Table with patterns to verify. (Default `patterns`).
- **intThreads** (*int*) – Number of threads for desktop mode. (Default 1).
- **useSGE** (*boolean*) – `True` if patterns should be generated on SGE computer cluster, `False` for desktop mode. (Default `False``).

`BrEPSdb.verification.`**verify_pattern**(*tplData*)
Verifies a pattern against all sequences.

Checks every pattern against all sequences in the `sequences` table. Calculates true positives (tp), false positives (fp) and false negatives (fn) to rank the patterns. The correstonding pattern gets updated values for tp, fp, and fn.

Parameters **tplData** (*tuple*) – Tuple containing two values. Table name of the pattern and pattern id to verify.

2.7 createdb.py

This module creates the final databases in MySQL and SQLite format. Additionally the patterns get an annotation with EC numbers, accessions, and proposed function based on a consensus EC number.

`BrEPSdb.createdb.`**count_positions**(*strPS*)
Calculates the number of positions in a pattern.

Parameters **strPS** (*string*) – Pattern in PROSITE format.

Returns Number of positions.

Return type int

`BrEPSdb.createdb.`**get_consensus**(*strEcs*)
Creates a consensus EC number from multiple EC numbers.

The lowest EC number class level where all ECs match. Result can be a partial EC number (like 1.1.1.-).

Parameters **strEcs** (*str*) – Multiple EC numbers separated with , (comma), multiple domains separated by & (ampersand).

Returns A list with all consensus EC numbers.

Return type list

BrEPSdb.createdb.**match_ecs**(*strEc1*, *strEc2*)
Creates a partial match of two EC numbers.

Example: 1.1.1.1 and 1.1.1.2 create 1.1.1

Parameters

- **strEc1** (*str*) – The first EC number.
- **strEc2** (*str*) – The second EC number.

Returns the overlapping part of both EC numbers or None if no match.

Return type str

BrEPSdb.createdb.**pattern_len**(*strPS*)
Calculates the minimum and maximum length of a pattern.

Parameters **strPS** (*string*) – Pattern in PROSITE format.

Returns Tuple with two values for minimu and maximum pattern length.

Return type tuple

BrEPSdb.createdb.**prepare_data**()
Prepares pattern data for final database creation.

Returns Tuple containing two lists of tuples with annotation and pattern data for saving.

Return type tuple

BrEPSdb.createdb.**prepare_db**()
Prepares the MySQL and SQLite database.

Prepares SQLite and MySQL databases for final release.

BrEPSdb.createdb.**save**(*strTable*, *lstData*, *lstFields*)
Saves data to MySQL and SQLite.

Parameters

- **strTable** (*str*) – Table name to save in.
- **lstData** (*list*) – List of tuples with data to save.
- **lstFields** (*list*) – List of field names for query generation.

BrEPSdb.createdb.**subpatterns**(*strPS*, *intGaplen=100*)
Generates sub-patterns for a given pattern.

The sub-patterns are calculates based on a given gap length for variable positions in the pattern. A sub-pattern needs at least 8 positions (just like a regular pattern).

Note: Sub-patterns are currently not used by BrEPS 2.0.

Parameters

- **strPS** (*str*) – Pattern in PROSITE format.
- **intGaplen** (*int*) – Minimum length of a gap to cut the pattern (Default 100).

Returns A list of sub-patterns in PROSITE format for the given pattern.

Return type list

CHAPTER

THREE

BREPSLIB

3.1 brepslib.py

This library contains regular expressions, functions and classes used by multiple BrEPS modules.

class BrEPSdb.brepslib.**Dbase**(*strType*, *strDB*, *strHost=None*, *strUser=None*, *strPass=None*)
Database class for both MySQL and SQLite usage.

On creation the strType can be selected from MySQL and SQLite to select a database type. All functions are created just like the provided by MySQLdb or sqlite3. MySQL placeholders %s are automatically replaced for SQLite. Before every execution the connection is checked to avoid mysql.OperationalError exceptions. The returned instance can be directly used for execution. Also the instance needs to be closed properly.

Only non-standard methods are documented!

strType
str – Type of database connection (mysql or sqlite).

strDB
str – Database name for MySQL, path to file for SQLite.

strHost
str – Host for the MySQL connection.

strUser
str – Username for the MySQL connection.

strPass
str – Password for the MySQL connection.

BrEPSdb.brepslib.**complexity**(*strPS*)
Calculates the complexity from a PROSITE pattern.

The pattern complexity is calculated for every amino acid by the following rules:

- A conserved amino acid has complexity 0.0.
- Set of n aminoacids has complexity of n/20.
- Wildcard positions (x) have complexity of 1.0.

The sum of all complexities will be divided by the maximum length of the pattern.

Parameters **strPS** (*str*) – Patten in PROSITE format.

Returns

Complexity ranging from 1.0 (complex) to 0.0 (not complex).

Return type float

BrEPSdb.brepslib.**ps2consensus**(*strProsite*)
Converts PROSITE patterns to consensus representation.

Parameters **strProsite** (*str*) – The PROSITE pattern.

Returns Consensus representation of the pattern.

Return type str

BrEPSdb.brepslib.**ps2regex**(*strProsite*)
Converts PROSITE patterns to regular expressions.

Parameters **strProsite** (*str*) – The PROSITE pattern.

Returns Regular expression representation of the pattern.

Return type str

BrEPSdb.brepslib.**subpattern**(*strPS*)
Creates a low complexity subpattern for pattern pre checks.

Parameters **strPS** (*str*) – Pattern in PROSITE pattern format.

Returns Subpattern of strPS in PROSITE format.

Return type str

PYTHON MODULE INDEX

b

INDEX

T

V