KripoDB Documentation

Release 3.0.0

Stefan Verhoeven

Mar 30, 2018

Contents

For installation and usage see https://github.com/3D-e-Chem/kripodb/blob/master/README.md

CHAPTER 1

Data update

The Kripo data can be updated in 2 ways:

1.1 Baseline update

Contents

The Kripo data set is generated from scratch every year or when algorithms change.

1.1.1 1. Create staging directory

Setup path with update scripts using:

```
export SCRIPTS=$PWD/../kripodb/update_scripts
```

Create a new directory:

```
mkdir staging
cd ..
```

1.1.2 2. Create sub-pocket pharmacophore fingerprints

Use directory listing of new pdb files as input:

```
ls $PDBS_ADDED_DIR | pdblist2fps_final_local.py
```

Todo: Too slow when run on single cpu. Chunkify input, run in parallel and merge results

1.1.3 3. Create fragment information

1. Fragment shelve

Where the fragment came from is stored in a Python shelve file. It can be generated from the pharmacophore files using:

```
compiledDatabase.py
```

2. Fragment sdf

The data generated thus far contains the molblocks of the ligands and atom nrs of each fragment. The fragment molblocks can be generated into a fragment sdf file with:

```
fragid2sd.py > fragments.sd
```

3. Pharmacophores

The raw pharmacophores are stored in the FRAGMENT_PPHORES sub-directory. Each pocket has a *_pphore.sd.gz file which contains the pharmacophore points of the whole pocket and a *_pphores.txt file which contains the indexes of pharmacophore points for each sub pocket or fragment. The raw pharmacophores need to be added to the pharmacophores datafile with:

```
kripodb pharmacophores add FRAGMENT_PPHORES pharmacophores.h5
```

1.1.4 4. Add new fragment information to fragment sqlite db

The following commands add the fragment shelve and sdf to the fragments database:

```
cp ../current/fragments.sqlite .
kripodb fragments shelve fragments.shelve fragments.sqlite
kripodb fragments sdf fragments.sd fragments.sqlite
```

Step 4 and 5 can be submitted to scheduler with:

```
jid_db=$(sbatch --parsable -n 1 -J db_append $SCRIPTS/db_append.sh)
```

1.1.5 5. Populate PDB metadata in fragments database

The following command will updated the PDB metadata to fragments database:

```
kripodb fragments pdb fragments.sqlite
```

1.1.6 6. Check no fragments are duplicated

The similarity matrix can not handle duplicates. It will result in addition of scores:

```
jid_dups=$(sbatch --parsable -n 1 -J check_dups --dependency=afterok:$jid_db $SCRIPTS/
↪baseline_duplicates.sh)
```

1.1.7 7. Calculate similarity scores between fingerprints

The similarities between fingerprints can be calculated with:

```
all_chunks=$(ls *fp.gz |wc -l)
jid_fpunzip=$(sbatch --parsable -n $all_chunks -J fpunzip --dependency=afterok:$jid_
↪dups $SCRIPTS/baseline_fpunzip.sh)
nr_chunks="$(($all_chunks * $all_chunks / 2 - $all_chunks))"
jid_fpneigh=$(sbatch --parsable -n $nr_chunks -J fpneigh --dependency=afterok:$jid_
↪fpunzip $SCRIPTS/baseline_similarities.sh)
jid_fpzip=$(sbatch --parsable -n $all_chunks -J fpzip --dependency=afterok:$jid_
↪fpneigh $SCRIPTS/baseline_fpzip.sh)
jid_merge_matrices=$(sbatch --parsable -n 1 -J merge_matrices --dependency=afterok:
↪$jid_fpneigh $SCRIPTS/baseline_merge_similarities.sh)
```

To prevent duplicates similarities of a chunk against itself should ignore the upper triangle.

Todo: Don't fpneigh run sequentially but submit to batch queue system and run in parallel

1.1.8 8. Convert pairs file into dense similarity matrix

Tip: Converting the pairs file into a dense matrix goes quicker with more memory.

The following commands converts the pairs into a compressed dense matrix:

```
jid_compress_matrix=$(sbatch --parsable -n 1 -J compress_matrix --dependency=afterok:
↪$jid_merge_matrices $SCRIPTS/freeze_similarities.sh)
```

The output of this step is ready to be served as a webservice using the *kripodb serve* command.

1.1.9 9. Switch staging to current

The webserver and webservice are configure to look in the *current* directory for files.

The staging can be made current with the following commands:

```
mv current old
mv staging current
```

1.1.10 10.0 Update web service

The webservice running at http://3d-e-chem.vu-compmedchem.nl/kripodb must be updated with the new datafiles.

The following files must copied to the server

- fragments.sqlite
- pharmacophores.h5
- similarities.packedfrozen.h5

The webservice must be restarted.

To show how up to date the webservice is the release date of the latest PDB is stored in *version.txt* which can be reached at http://3d-e-chem.vu-compmedchem.nl/kripodb/version.txt The content *version.txt* must be updated.

1.2 Incremental update

Contents

The Kripo data set can be incrementally updated with new PDB entries.

1.2.1 1. Create staging directory

Setup path with update scripts using:

```
export SCRIPTS=$PWD/../kripodb/update_scripts
```

Create a new directory:

```
mkdir staging
cd ..
```

1.2.2 2. Create sub-pocket pharmacophore fingerprints

The *ids.txt* file must contain a list of PDB identifiers which have not been processed before. It can be fetched from https://www.rcsb.org/.

Adjust the PDB save location in the *singleprocess.py* script to the staging directory.

Run the following command to generate fragments/pharmacophores/fingerprints for each PDB listed in *ids.txt*:

```
python singleprocess.py
```

1.2.3 3. Create fragment information

1. Fragment shelve

Where the fragment came from is stored in a Python shelve file. It can be generated from the pharmacophore files using:

```
compiledDatabase.py
```

2. Fragment sdf

The data generated thus far contains the molblocks of the ligands and atom nrs of each fragment. The fragment molblocks can be generated into a fragment sdf file with:

```
fragid2sd.py fragments.shelve > fragments.sd
```

3. Pharmacophores

The raw pharmacophores are stored in the FRAGMENT_PPHORES sub-directory. Each pocket has a *_pphore.sd.gz file which contains the pharmacophore points of the whole pocket and a *_pphores.txt file which contains the indexes of pharmacophore points for each sub pocket or fragment. The raw pharmacophores of the update can be added to the existing pharmacophores datafile with:

```
cp ../current/pharmacophores.h5 .
kripodb pharmacophores add FRAGMENT_PPHORES pharmacophores.h5
```

1.2.4 4. Add new fragment information to fragment sqlite db

The following commands add the fragment shelve and sdf to the fragments database:

```
cp ../current/fragments.sqlite .
kripodb fragments shelve fragments.shelve fragments.sqlite
kripodb fragments sdf fragments.sd fragments.sqlite
```

Step 4 and 5 can be submitted to scheduler with:

```
jid_db=$(sbatch --parsable -n 1 -J db_append $SCRIPTS/db_append.sh)
```

1.2.5 5. Populate PDB metadata in fragments database

The following command will updated the PDB metadata to fragments database:

```
kripodb fragments pdb fragments.sqlite
```

1.2.6 6. Check no fragments are duplicated

The similarity matrix can not handle duplicates. It will result in addition of scores:

```
jid_dups=$(sbatch --parsable -n 1 -J check_dups --dependency=afterok:$jid_db $SCRIPTS/
↪incremental_duplicates.sh)
```

1.2.7 7. Calculate similarity scores between fingerprints

The similarities between the new and existing fingerprints and between new fingerprints themselves can be calculated with:

```
current_chunks=$(ls ../current/*fp.gz |wc -l)
all_chunks=$(($current_chunks + 1))
jid_fpneigh=$(sbatch --parsable -n $all_chunks -J fpneigh --dependency=afterok:$jid_
↪dups $SCRIPTS/incremental_similarities.sh)
jid_merge_matrices=$(sbatch --parsable -n 1 -J merge_matrices --dependency=afterok:
↪$jid_fpneigh $SCRIPTS/incremental_merge_similarities.sh)
```

1.2.8 8. Convert pairs file into dense similarity matrix

Note: Converting the pairs file into a dense matrix goes quicker with more memory.

The frame size (-f) should be as big as possible, 100000000 requires 6Gb RAM.

The following commands converts the pairs into a compressed dense matrix:

```
jid_compress_matrix=$(sbatch --parsable -n 1 -J compress_matrix --dependency=afterok:
↪$jid_merge_matrices $SCRIPTS/freeze_similarities.sh)
```

The output of this step is ready used to find similar fragments, using either the webservice with the *kripodb serve* command or with the *kripodb similarities similar* command directly.

1.2.9 9. Switch staging to current

The webserver and webservice are configure to look in the *current* directory for files.

The current and new pharmacophores need to be combined:

```
mv staging/FRAGMENT_PPHORES staging/FRAGMENT_PPHORES.new
rsync -a current/FRAGMENT_PPHORES staging/FRAGMENT_PPHORES
rm -r staging/FRAGMENT_PPHORES.new
```

Todo: rsync of current/FRAGMENT_PPHORES to destination, maybe too slow due large number of files. Switch to move old pharmacohores and rsync new pharmacophores into it when needed.

The current and new fingerprints need to be combined:

```
cp -n current/*.fp.gz staging/
```

The staging can be made current with the following commands:

```
mv current old && mv staging current
```

9.1 Merge fingerprint files (optional)

To keep the number of files to a minimum it is advised to merge the fingerprint files from incremental updates of a year.

The incremental fingerprint files are named like *out.<year><week>.fp.gz*, to generate kripo_fingerprints_<year>_fp.gz run:

```
sbatch --parsable -n 1 -J merge_fp $SCRIPTS/incremental_merge_fp.sh <year>
```

1.2.10 10.0 Update web service

The webservice running at http://3d-e-chem.vu-compmedchem.nl/kripodb must be updated with the new datafiles.

The following files must copied to the server

- fragments.sqlite

- pharmacophores.h5
- similarities.packedfrozen.h5

The webservice must be restarted.

To show how up to date the webservice is the release date of the latest PDB is stored in *version.txt* which can be reached at http://3d-e-chem.vu-compmedchem.nl/kripodb/version.txt The content *version.txt* must be updated.

1.3 Steps

Overview of steps involved in updating Kripo:

1. Create staging directory
2. Create sub-pocket pharmacophore fingerprints
3. Create fragment information
4. Add new fragment information to fragment sqlite db
5. Populate PDB metadata in fragments database
6. Check no fragments are duplicated
7. Calculate similarity scores between fingerprints
8. Convert pairs file into dense similarity matrix
9. Switch staging to current
10. Update web service

Note: Steps 2 through 3 require undisclosed scripts or https://github.com/3D-e-Chem/kripo

Note: Steps 4 and 6 through 7 can be done using the KripoDB Python library.

Todo: Remove Kripo fragment/fingerprints of obsolete PDBs (ftp://ftp.wwpdb.org/pub/pdb/data/status/obsolete.dat)

1.4 Disk layout

Directories for Kripo:

- **current/**, directory which holds current dataset
- **staging/**, which is used to compute new items and combine new and old items.
- **old/**, which is used as a backup containing the previous update.

Files and directories for a data set (inside current, staging and old directories):

- **pharmacophores.h5**, pharmacophores database file
- **out.fp.sqlite**, fingerprints file
- **fragments.sqlite**, fragment information database file

- **similarities.h5**, similarities as pairs table
- **similarities.packedfrozen.h5**, similarities as dense matrix

Input directories:

- **$PDBS_ADDED_DIR**, directory containing new PDB files to be processed

1.5 Requirements

- Slurm batch scheduler
- KripoDB and it's dependencies installed and in path
- Posix filesystem, NFS of Virtualbox share do not accept writing of hdf5 or sqlite files

CHAPTER 2

DiVE visualization

DiVE homepage at https://github.com/NLeSC/DiVE

The Kripo similarity matrix can be embedded to 2D or 3D using largevis and then visualized using DiVE.

Steps

1. LargeVis input file from Kripo similarity matrix
2. Perform embedding using LargeVis
3. Generate DiVE metadata datafiles
4. Create DiVE input file

Input datasets

1. only fragment1 or whole unfragmented ligands
2. all fragments
3. only gpcr frag1
4. only kinase frag1
5. only gpcr and kinase frag1

Output datasets

1. 2D
2. 3D

2.1 1. LargeVis input file from Kripo similarity matrix

Dump the similarity matrix to csv of *frag1 fragments:

```
kripodb similarities export --no_header --frag1 similarities.h5 similarities.frag1.txt
```

2.1.1 Similarities between GPCR pdb entries

Use the GPCRDB web service to fetch a list of PDB codes which contain GPCR proteins:

```
curl -X GET --header 'Accept: application/json' 'http://gpcrdb.org/services/structure/
↪' | jq  -r '.[] | .pdb_code' > pdb.gpcr.txt
```

Dump the similarity matrix to csv:

```
kripodb similarities export --no_header --frag1 --pdb pdb.gpcr.txt similarities.h5␣
↪similarities.frag1.gpcr.txt
```

2.1.2 Similarities between GPCR and Kinase pdb entries

Use the KLIFS KNIME nodes to create a file with of PDB codes of Kinases called *pdb.kinase.txt*.

Dump the similarity matrix to csv:

```
cat pdb.gpcr.txt pdb.kinase.txt > pdb.gpcr.kinase.txt
kripodb similarities export --no_header --frag1 --pdb pdb.gpcr.kinase.txt␣
↪similarities.h5 similarities.frag1.gpcr.kinase.txt
```

2.2 2. Perform embedding using LargeVis

Get or compile LargeVis binaries from https://github.com/lferry007/LargeVis

Compile using miniconda:

```
conda install gsl gcc
cd LargeVis/Linux
c++ LargeVis.cpp main.cpp -o LargeVis -lm -pthread -lgsl -lgslcblas -Ofast -Wl,-rpath,
↪$CONDA_PREFIX/lib -march=native -ffast-math
cp LargeVis $CONDA_PREFIX/bin/
```

Then embed frag1 similarity matrix in 3D with:

```
LargeVis -fea 0 -outdim 3 -threads $(nproc) -input similarities.frag1.txt -output␣
↪largevis.frag1.3d.txt
```

Then embed frag1 similarity matrix in 2D with:

```
LargeVis -fea 0 -outdim 2 -threads $(nproc) -input similarities.frag1.txt -output␣
↪largevis.frag1.2d.txt
```

Then embed similarity matrix in 3D with:

```
LargeVis -fea 0 -outdim 3 -threads $(nproc) -input similarities.txt -output largevis.
↪3d.txt
```

Then embed similarity matrix in 2D with:

```
LargeVis -fea 0 -outdim 2 -threads $(nproc) -input similarities.txt -output largevis.
↪2d.txt
```

The *kripo export* in step 1 and the LargeVis command can be submitted to scheduler with:

```
sbatch -n 1 $SCRIPTS/dive_frag1.sh
sbatch -n 1 $SCRIPTS/dive_frag1_gpcr_kinase.sh
```

2.3 3. Generate DiVE metadata datafiles

Command to generate properties files:

```
wget -O uniprot.txt 'http://www.uniprot.org/uniprot/?query=database:pdb&format=tab&
↪columns=id,genes(PREFERRED),families,database(PDB)'
kripodb dive export --pdbtags pdb.gpcr.txt --pdbtags pdb.kinase.txt fragments.sqlite␣
↪uniprot.txt
```

Will generate in current working directory the following files:

- kripo.props.txt
- kripo.propnames.txt

2.4 4. Create DiVE input file

DiVE has a script which can combine the LargeVis coordinates together with metadata. Download the MakeVizDataWithProperMetadata.py script from https://github.com/NLeSC/DiVE/blob/master/scripts_prepareData/MakeVizDataWithProperMetadata.py

For more information about the script see https://github.com/NLeSC/DiVE#from-output-of-largevis-to-input-of-dive .

Example command to generate new DiVE input file:

```
python MakeVizDataWithProperMetadata.py -coord largevis2.similarities.frag1.gpcr.
↪kinase.txt -metadata kripo.props.txt -np kripo.propnames.txt -json largevis2.
↪similarities.frag1.gpcr.kinase.json -dir .
```

The generated file (largevis2.similarities.frag1.gpcr.kinase.json) can be uploaded at https://nlesc.github.io/DiVE/ to visualize.

CHAPTER 3

API

3.1 kripodb.canned

Module with functions which use pandas DataFrame as input and output.

For using Kripo data files inside KNIME (http://www.knime.org)

exception kripodb.canned.**IncompleteHits**(*absent_identifiers*, *hits*)

kripodb.canned.**fragments_by_id**(*fragment_ids*, *fragments_db_filename_or_url*, *prefix=''*)

Retrieve fragments based on fragment identifier.

Parameters

- **fragment_ids** (*List[str]*) – List of fragment identifiers
- **fragments_db_filename_or_url** (*str*) – Filename of fragments db or base url of kripodb webservice
- **prefix** (*str*) – Prefix for output columns

Examples

Fetch fragments of '2n2k_MTN_frag1' fragment identifier

```
>>> from kripodb.canned import fragments_by_id
>>> fragment_ids = pd.Series(['2n2k_MTN_frag1'])
>>> fragments = fragments_by_id(fragment_ids, 'data/fragments.sqlite')
>>> len(fragments)
1
```

Retrieved from web service instead of local fragments db file. Make sure the web service is running, for example by *kripodb serve data/similarities.h5 data/fragments.sqlite data/pharmacophores.h5*.

```
>>> fragments = fragments_by_id(fragment_ids,, 'http://localhost:8084/kripo')
>>> len(fragments)
1
```

Returns Data frame with fragment information

Return type pandas.DataFrame

Raises `IncompleteFragments` – When one or more of the identifiers could not be found.

`kripodb.canned.`**`fragments_by_pdb_codes`**(*pdb_codes*, *fragments_db_filename_or_url*, *prefix=''*)

Retrieve fragments based on PDB codes.

See http://www.rcsb.org/pdb/ for PDB structures.

Parameters

- **`pdb_codes`** (*`List[str]`*) – List of PDB codes
- **`fragments_db_filename_or_url`** (*`str`*) – Filename of fragments db or base url of kripodb webservice
- **`prefix`** (*`str`*) – Prefix for output columns

Examples

Fetch fragments of '2n2k' PDB code

```
>>> from kripodb.canned import fragments_by_pdb_codes
>>> pdb_codes = pd.Series(['2n2k'])
>>> fragments = fragments_by_pdb_codes(pdb_codes, 'data/fragments.sqlite')
>>> len(fragments)
3
```

Retrieved from web service instead of local fragments db file. Make sure the web service is running, for example by *kripodb serve data/similarities.h5 data/fragments.sqlite data/pharmacophores.h5*.

```
>>> fragments = fragments_by_pdb_codes(pdb_codes, 'http://localhost:8084/kripo')
>>> len(fragments)
3
```

Returns Data frame with fragment information

Return type pandas.DataFrame

Raises `IncompleteFragments` – When one or more of the identifiers could not be found.

`kripodb.canned.`**`pharmacophores_by_id`**(*fragment_ids*, *pharmacophores_db_filename_or_url*)

Fetch pharmacophore points by fragment identifiers

Parameters

- **`fragment_ids`** (*`pd.Series`*) – List of fragment identifiers
- **`pharmacophores_db_filename_or_url`** – Filename of pharmacophores db or base url of kripodb webservice

Returns

Pandas series with pharmacophores as string in phar format. Fragment without pharmacophore will return None

Return type pandas.Series

Examples

Fragments similar to '3j7u_NDP_frag24' fragment.

```
>>> from kripodb.canned import pharmacophores_by_id
>>> fragment_ids = pd.Series(['2n2k_MTN_frag1'], ['Row0'])
>>> pharmacophores = pharmacophores_by_id(fragment_ids, 'data/pharmacophores.h5')
>>> len(pharmacophores)
1
```

Retrieved from web service instead of local pharmacophores db file. Make sure the web service is running, for example by *kripodb serve data/similarities.h5 data/fragments.sqlite data/pharmacophores.h5*.

```
>>> pharmacophores = pharmacophores_by_id(fragment_ids, 'http://localhost:8084/
↪kripo')
>>> len(pharmacophores)
1
```

kripodb.canned.**similarities**(*queries*, *similarity_matrix_filename_or_url*, *cutoff*, *limit=1000*)

Find similar fragments to queries based on similarity matrix.

Parameters

- **queries** (*List[str]*) – Query fragment identifiers
- **similarity_matrix_filename_or_url** (*str*) – Filename of similarity matrix file or base url of kripodb webservice
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits for each query. Default is 1000. Use is None for no limit.

Examples

Fragments similar to '3j7u_NDP_frag24' fragment.

```
>>> import pandas as pd
>>> from kripodb.canned import similarities
>>> queries = pd.Series(['3j7u_NDP_frag24'])
>>> hits = similarities(queries, 'data/similaritys.h5', 0.55)
>>> len(hits)
11
```

Retrieved from web service instead of local similarity matrix file. Make sure the web service is running, for example by *kripodb serve data/similarities.h5 data/fragments.sqlite data/pharmacophores.h5*.

```
>>> hits = similarities(queries, 'http://localhost:8084/kripo', 0.55)
>>> len(hits)
11
```

Returns Data frame with query_fragment_id, hit_frag_id and score columns

Return type pandas.DataFrame

Raises *IncompleteHits* – When one or more of the identifiers could not be found.

3.2 kripodb.db

Fragments and fingerprints sqlite based data storage.

Registers *BitMap* and *molblockgz* data types in sqlite.

class kripodb.db.**FastInserter**(*cursor*)

Use with to make inserting faster, but less safe

By setting journal mode to WAL and turn synchronous off.

Parameters **cursor** (*sqlite3.Cursor*) – Sqlite cursor

Examples

```
>>> with FastInserter(cursor):
        cursor.executemany('INSERT INTO table VALUES (?), rows))
```

class kripodb.db.**FingerprintsDb**(*filename*)

Fingerprints database

as_dict(*number_of_bits=None*)

Returns a dict-like object to query and alter fingerprints db

Parameters **number_of_bits** (*Optional[int]*) – Number of bits that all fingerprints have

Returns BitMapDict

create_tables()

Abstract method which is called after connecting to database so tables can be created.

Use *CREATE TABLE IF NOT EXISTS ...* in method to prevent duplicate create errors.

class kripodb.db.**FragmentsDb**(*filename*)

Fragments database

add_fragment(*frag_id*, *pdb_code*, *prot_chain*, *het_code*, *frag_nr*, *atom_codes*, *hash_code*, *het_chain*, *het_seq_nr*, *nr_r_groups*)

Add fragment to database

Parameters

- **frag_id** (*str*) – Fragment identifier
- **pdb_code** (*str*) – Protein databank identifier
- **prot_chain** (*str*) – Major chain of pdb on which pharmacophore is based
- **het_code** (*str*) – Ligand/Hetero code
- **frag_nr** (*int*) – Fragment number, whole ligand has number 1, fragments are >1
- **atom_codes** (*str*) – Comma separated list of HETATOM atom names which make up the fragment (hydrogens are excluded)
- **hash_code** (*str*) – Unique identifier for fragment

- **het_chain** (*str*) – Chain ligand is part of
- **het_seq_nr** (*int*) – Residue sequence number of ligand the fragment is a part of
- **nr_r_groups** (*int*) – Number of R groups in fragment

add_fragments_from_shelve(*myshelve*, *skipdups=False*)
Adds fragments from shelve to fragments table.

Also creates index on pdb_code column.

Parameters

- **myshelve** (*Dict[Fragment]*) – Dictionary with fragment identifier as key and fragment as value.
- **skipdups** (*bool*) – Skip duplicates, instead of dieing one first duplicate

add_molecule(*mol*)
Adds molecule to molecules table

Uses the name of the molecule as the primary key.

Parameters **mol** (*rdkit.Chem.AllChem.Mol*) – the rdkit molecule

add_molecules(*mols*)
Adds molecules to to molecules table.

Parameters **mols** (*list[rdkit.Chem.Mol]*) – List of molecules

add_pdbs(*pdbs*)
Adds pdb meta data to to pdbs table.

Parameters **pdbs** (*Iterable[Dict]*) – List of pdb meta data

by_pdb_code(*pdb_code*)
Retrieve fragments which are part of a PDB structure.

Parameters **pdb_code** (*str*) – PDB code

Returns List of fragments

Return type List[Fragment]

Raises LookupError – When pdb_code could not be found

create_tables()
Create tables if they don't exist

id2label()
Lookup table of fragments from an number to a label.

Returns SqliteDict

is_ligand_stored(*pdb_code*, *het_code*)
Check whether ligand is already in database

Parameters

- **pdb_code** (*str*) – Protein databank identifier
- **het_code** (*str*) – Ligand/hetero identifier

Returns bool

label2id()
Lookup table of fragments from an label to a number.

Returns SqliteDict

class kripodb.db.**IntbitsetDict**(*db*, *number_of_bits=None*)
Dictionary of BitMaps with sqlite3 backend.

Parameters

- **db** (FingerprintsDb) – Fingerprints db
- **number_of_bits** (*int*) – Number of bits

number_of_bits
int – Number of bits the bitsets consist of

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

class kripodb.db.**SqliteDb**(*filename*)
Wrapper around a sqlite database connection

Database is created if it does not exist.

Parameters **filename** (*str*) – Sqlite filename

connection
sqlite3.Connection – Sqlite connection

cursor
sqlite3.Cursor – Sqlite cursor

close()
Close database

commit()
Commit pending changes

create_tables()
Abstract method which is called after connecting to database so tables can be created.

Use *CREATE TABLE IF NOT EXISTS ...* in method to prevent duplicate create errors.

class kripodb.db.**SqliteDict**(*connection*, *table_name*, *key_column*, *value_column*)
Dict-like object of 2 columns of a sqlite table.

Can be used to query and alter the table.

Parameters

- **connection** (*sqlite3.Connection*) – Sqlite connection
- **table_name** (*str*) – Table name
- **key_column** (*str*) – Column name used as key
- **value_column** (*str*) – Column name used as value

connection
sqlite3.Connection – Sqlite connection

cursor
sqlite3.Cursor – Sqlite cursor

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iteritems_startswith(*prefix*)
item iterator over keys with prefix

Parameters **prefix** (*str*) – Prefix of key

Examples

All items with key starting with letter 'a' are returned.

```
>>> for frag_id, fragment in fragments.iteritems_startswith('a'):
        # do something with frag_id and fragment
```

Returns List[Tuple[key, value]]

itervalues() → an iterator over the values of D

materialize()
Fetches all kev/value pairs from the sqlite database.

Useful when dictionary is iterated multiple times and the cost of fetching is to high.

Returns Dictionary with all kev/value pairs

Return type Dict

values() → list of D's values

kripodb.db.**adapt_BitMap**(*ibs*)
Convert BitMap to it's serialized format

Parameters **ibs** (*BitMap*) – bitset

Examples

Serialize BitMap

```
>>> adapt_BitMap(BitMap([1, 2, 3, 4]))
'xc@\{ð\'
```

Returns serialized BitMap

Return type str

kripodb.db.**adapt_molblockgz**(*mol*)
Convert RDKit molecule to compressed molblock

Parameters **mol** (*rdkit.Chem.Mol*) – molecule

Returns Compressed molblock

Return type str

kripodb.db.**convert_BitMap**(*s*)
Convert serialized BitMap to BitMap

Parameters **s** (*str*) – serialized BitMap

Examples

Deserialize BitMap

```
>>> ibs = convert_BitMap('xc@\{ð\')
BitMap([1, 2, 3, 4])
```

Returns bitset

Return type BitMap

kripodb.db.**convert_molblockgz**(*molgz*)
Convert compressed molblock to RDKit molecule

Parameters **molgz** – (str) zlib compressed molblock

Returns molecule

Return type rdkit.Chem.Mol

3.3 kripodb.dive

kripodb.dive.**dense_dump**(*inputfile*, *outputfile*, *frag1only*)
Dump dense matrix with zeros included

Parameters

- **inputfile** (*str*) – Filename of dense similarity matrix
- **outputfile** (*file*) – Writeable file object
- **frag1only** (*bool*) – Only dump frag1 fragments

Returns:

kripodb.dive.**dense_dump_iter**(*matrix*, *frag1only*)
Iterate dense matrix with zeros

Parameters

- **matrix** (FrozenSimilarityMatrix) – Dense similarity matrix
- **frag1only** (*bool*) – True to iterate over *frag1 only

Yields *(str, str, float)* – Fragment label pair and score

kripodb.dive.**dive_export**(*fragmentsdb*, *uniprot_annot*, *pdbtags*, *propnames*, *props*)
Writes metdata props for DiVE visualization

Parameters

- **fragmentsdb** (*str*) – Filename fo fragments db file
- **uniprot_annot** (*file*) – Readable file object with uniprot gene and family mapping as tsv
- **pdbtags** (*list*) – List of readable file objects to tag pdb by filename
- **propnames** (*file*) – Writable file object to write prop names to
- **props** (*file*) – Writeable file object to write props to

kripodb.dive.**dive_sphere**(*inputfile*, *outputfile*, *onlyfrag1*)
Export fragments as DiVE formatted sphere

Parameters

- **inputfile** (*str*) – fragments db input file
- **outputfile** (*file*) – fragments dive output file
- **onlyfrag1** (*bool*) – Only *_frag1

3.4 kripodb.frozen

Similarity matrix using pytables carray

class kripodb.frozen.**FrozenSimilarityMatrix**(*filename*, *mode='r'*, ***kwargs*)
Frozen similarities matrix

Can retrieve whole column of a specific row fairly quickly. Store as compressed dense matrix. Due to compression the zeros use up little space.

Warning! Can not be enlarged.

Compared find performance FrozenSimilarityMatrix with SimilarityMatrix:

```
>>> from kripodb.db import FragmentsDb
>>> db = FragmentsDb('data/feb2016/Kripo20151223.sqlite')
>>> ids = [v[0] for v in db.cursor.execute('SELECT frag_id FROM fragments ORDER␣
↪BY RANDOM() LIMIT 20')]
>>> from kripodb.frozen import FrozenSimilarityMatrix
>>> fdm = FrozenSimilarityMatrix('01-01_to_13-13.out.frozen.blosczlib.h5')
>>> from kripodb.hdf5 import SimilarityMatrix
>>> dm = SimilarityMatrix('data/feb2016/01-01_to_13-13.out.h5', cache_labels=True)
>>> %timeit list(dm.find(ids[0], 0.45, None))
```

... 1 loop, best of 3: 1.96 s per loop >>> %timeit list(fdm.find(ids[0], 0.45, None)) ... The slowest run took 6.21 times longer than the fastest. This could mean that an intermediate result is being cached. ... 10 loops, best of 3: 19.3 ms per loop >>> ids = [v[0] for v in db.cursor.execute('SELECT frag_id FROM fragments ORDER BY RANDOM() LIMIT 20')] >>> %timeit -n1 [list(fdm.find(v, 0.45, None)) for v in ids] ... 1 loop, best of 3: 677 ms per loop >>> %timeit -n1 [list(dm.find(v, 0.45, None)) for v in ids] ... 1 loop, best of 3: 29.7 s per loop

Parameters

- **filename** (*str*) – File name of hdf5 file to write or read similarity matrix from
- **mode** (*str*) – Can be 'r' for reading or 'w' for writing
- ****kwargs** – Passed though to tables.open_file()

h5file
tables.File – Object representing an open hdf5 file

scores
tables.CArray – HDF5 Table that contains matrix

labels
tables.CArray – Table to look up label of fragment by id or id of fragment by label

close()
Closes the hdf5file

count (*frame_size=None*, *raw_score=False*, *lower_triangle=False*)
Count occurrences of each score

Only scores are counted of the upper triangle or lower triangle. Zero scores are skipped.

Parameters

- **frame_size** (*int*) – Dummy argument to force same interface for thawed and frozen matrix
- **raw_score** (*bool*) – When true return raw int16 score else fraction score
- **lower_triangle** (*bool*) – When true return scores from lower triangle else return scores from upper triangle

Returns Score and number of occurrences

Return type Tuple[(str, int)]

find (*query*, *cutoff*, *limit=None*)
Find similar fragments to query.

Parameters

- **query** (*str*) – Query fragment identifier
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Returns Hit fragment identifier and similarity score

Return type list[tuple[str,float]]

from_array (*data*, *labels*)
Fill matrix from 2 dimensional array

Parameters

- **data** (*np.array*) – 2 dimensional square array with scores
- **labels** (*list*) – List of labels for each column and row index

from_pairs (*similarity_matrix*, *frame_size*, *limit=None*, *single_sided=False*)
Fills self with matrix which is stored in pairs.

Also known as COOrdinate format, the 'ijv' or 'triplet' format.

Parameters

- **similarity_matrix** (kripodb.hdf5.SimilarityMatrix) –
- **frame_size** (*int*) – Number of pairs to append in a single go
- **limit** (*int|None*) – Number of pairs to add, None for no limit, default is None.
- **single_sided** (*bool*) – If false add stored direction and reverse direction. Default is False.

time kripodb similarities freeze –limit 200000 -f 100000 data/feb2016/01-01_to_13-13.out.h5 percell.h5 47.2s time kripodb similarities freeze –limit 200000 -f 100000 data/feb2016/01-01_to_13-13.out.h5 coo.h5 0.2m - 2m6s .4m - 2m19s .8m - 2m33s 1.6m - 2m48s 3.2m - 3m4s 6.4m - 3m50s 12.8m - 4m59s 25.6m - 7m27s

to_pairs (*pairs*)
Copies labels and scores from self to pairs matrix.

Parameters **pairs** (SimilarityMatrix) –

to_pandas()
Pandas dataframe with labelled colums and rows.

Warning! Only use on matrices that fit in memory

Returns pd.DataFrame

3.5 kripodb.hdf5

Similarity matrix using hdf5 as storage backend.

class kripodb.hdf5.**AbstractSimpleTable**(*table*, *append_chunk_size=100000000*)
Abstract wrapper around a HDF5 table

Parameters

- **table** (*tables.Table*) – HDF5 table
- **append_chunk_size** (*int*) – Size of chunk to append in one go. Defaults to 1e8, which when table description is 10bytes will require 2Gb during append.

Attributes table (tables.Table): HDF5 table append_chunk_size (int): Number of rows to read from other table during append.

append(*other*)
Append rows of other table to self

Parameters **other** – Table of same type as self

class kripodb.hdf5.**LabelsLookup**(*h5file*, *expectedrows=0*)
Table to look up label of fragment by id or id of fragment by label

When table does not exist in h5file it is created.

Parameters

- **h5file** (*tables.File*) – Object representing an open hdf5 file
- **expectedrows** (*int*) – Expected number of pairs to be added. Required when similarity matrix is opened in write mode, helps optimize storage

by_id(*frag_id*)
Look up label of fragment by id

Parameters **frag_id** (*int*) – Fragment identifier

Raises IndexError – When id of fragment is not found

Returns Label of fragment

Return type str

by_label(*label*)
Look up id of fragment by label

Parameters **label** (*str*) – Fragment label

Raises IndexError – When label of fragment is not found

Returns Fragment identifier

Return type int

by_labels(*labels*)
Look up ids of fragments by label

Parameters **labels** (*set[str]*) – Set of fragment labels

Raises IndexError – When label of fragment is not found

Returns Set of fragment identifiers

Return type set[int]

keep(*other*, *keep*)
Copy content of self to other and only keep given fragment identifiers

Parameters

- **other** (LabelsLookup) – Labels table to fill
- **keep** (*set[int]*) – Fragment identifiers to keep

label2ids()
Return whole table as a dictionary

Returns Dictionary with label as key and frag_id as value.

Return type dict

merge(*label2id*)
Merge label2id dict into self

When label does not exists an id is generated and the label/id is added. When label does exist the id of the label in self is kept.

Parameters **label2id** (*dict*) – Dictionary with fragment label as key and fragment identifier as value.

Returns Dictionary of label/id which where in label2id, but missing in self

Return type dict

skip(*other*, *skip*)
Copy content of self to other and skip given fragment identifiers

Parameters

- **other** (LabelsLookup) – Labels table to fill
- **skip** (*set[int]*) – Fragment identifiers to skip

update(*label2id*)
Update labels lookup by adding labels in label2id.

Parameters **label2id** (*dict*) – Dictionary with fragment label as key and fragment identifier as value.

class kripodb.hdf5.**PairsTable**(*h5file*, *expectedrows=0*)
Tabel to store similarity score of a pair of fragment fingerprints

When table does not exist in h5file it is created.

Parameters

- **h5file** (*tables.File*) – Object representing an open hdf5 file
- **expectedrows** (*int*) – Expected number of pairs to be added. Required when similarity matrix is opened in write mode, helps optimize storage

score_precision
int – Similarity score is a fraction, the score is converted to an int by multiplying it with the precision

full_matrix
bool – Matrix is filled above and below diagonal.

append(*other*)
Append rows of other table to self

Parameters **other** – Table of same type as self

count(*frame_size*, *raw_score=False*)
Count occurrences of each score

Parameters

- **frame_size** (*int*) – Size of matrix loaded each time. Larger requires more memory and smaller is slower.
- **raw_score** (*bool*) – Return raw int16 score or fraction score

Returns Score and number of occurrences

Return type Tuple[(str, int)]

find(*frag_id*, *cutoff*, *limit*)
Find fragment hits which has a similarity score with frag_id above cutoff.

Parameters

- **frag_id** (*int*) – query fragment identifier
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Returns Where first tuple value is hit fragment identifier and second value is similarity score

Return type List[Tuple]

keep(*other*, *keep*)
Copy pairs from self to other and keep given fragment identifiers and the identifiers they pair with.

Parameters

- **other** (PairsTable) – Pairs table to fill
- **keep** (*set[int]*) – Fragment identifiers to keep

Returns Fragment identifiers that have been copied to other

Return type set[int]

skip(*other*, *skip*)
Copy content from self to other and skip given fragment identifiers

Parameters

- **other** (PairsTable) – Pairs table to fill
- **skip** (*set[int]*) – Fragment identifiers to skip

update(*similarities_iter*, *label2id*)
Store pairs of fragment identifier with their similarity score

Parameters

- **similarities_iter** (*Iterator*) – Iterator which yields (label1, label2, similarity_score)
- **label2id** (*Dict*) – Lookup with fragment label as key and fragment identifier as value

class kripodb.hdf5.**SimilarityMatrix**(*filename*, *mode='r'*, *expectedpairrows=None*, *expectedlabelrows=None*, *cache_labels=False*, ***kwargs*)

Similarity matrix

Parameters

- **filename** (*str*) – File name of hdf5 file to write or read similarity matrix from
- **mode** (*str*) – Can be 'r' for reading or 'w' for writing
- **expectedpairrows** (*int*) – Expected number of pairs to be added. Required when similarity matrix is opened in write mode, helps optimize storage
- **expectedlabelrows** (*int*) – Expected number of labels to be added. Required when similarity matrix is opened in write mode, helps optimize storage
- **cache_labels** (*bool*) – Cache labels, speed up label lookups

h5file
tables.File – Object representing an open hdf5 file

pairs
PairsTable – HDF5 Table that contains pairs

labels
LabelsLookup – Table to look up label of fragment by id or id of fragment by label

append(*other*)
Append data from other similarity matrix to me

Parameters **other** (SimilarityMatrix) – Other similarity matrix

close()
Closes the hdf5file

count(*frame_size*, *raw_score=False*, *lower_triangle=False*)
Count occurrences of each score

Parameters

- **frame_size** (*int*) – Size of matrix loaded each time. Larger requires more memory and smaller is slower.
- **raw_score** (*bool*) – Return raw int16 score or fraction score
- **lower_triangle** (*bool*) – Dummy argument to force same interface for thawed and frozen matrix

Returns Score and number of occurrences

Return type (str, int)

find(*query*, *cutoff*, *limit=None*)
Find similar fragments to query.

Parameters

- **query** (*str*) – Query fragment identifier
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Yields *(str, float)* – Hit fragment idenfier and similarity score

keep(*other*, *keep*)
Copy content of self to other and only keep given fragment labels and the labels they pair with

Parameters

- **other** (SimilarityMatrix) – Writable matrix to fill
- **keep** (*set[str]*) – Fragment labels to keep

skip(*other*, *skip*)
Copy content of self to other and skip all given fragment labels

Parameters

- **other** (SimilarityMatrix) – Writable matrix to fill
- **skip** (*set[str]*) – Fragment labels to skip

update(*similarities_iter*, *label2id*)
Store pairs of fragment identifier with their similarity score and label 2 id lookup

Parameters

- **similarities_iter** (*iterator*) – Iterator which yields (label1, label2, similarity_score)
- **label2id** (*dict*) – Dictionary with fragment label as key and fragment identifier as value.

3.6 kripodb.makebits

Module to read/write fingerprints in Makebits file format

kripodb.makebits.**iter_file**(*infile*)
Reads Makebits formatted file Yields header first then tuples of identifier and BitMap object

Yields first header (format name, format version, number of bits, description), then tuples of the fingerprint identifier and an BitMap object

Parameters **infile** (*File*) – File object of Makebits formatted file to read

Examples

Read a file

```
>>> f = iter_file(open('fingerprints01.fp'))
>>> read_fp_size(next(f))
4
>>> {frag_id: fp for frag_id, fp in f}
{'id1': BitMap([1, 2, 3, 4])}
```

kripodb.makebits.**write_file**(*fp_size*, *bitsets*, *fn*)
Write makebits formatted file

Parameters

- **fp_size** (*int*) – Number of bits
- **bitsets** (*dict*) – Dict with fingerprint identifier as key and BitMap object as value

- **fn** (*File*) – File object to write to

Examples

Write a file

```
>>> write_file(4, {'id1': BitMap([1, 2, 3, 4])}, open('fingerprints01.fp', 'w'))
```

3.7 kripodb.modifiedtanimoto

Module to calculate modified tanimoto similarity

kripodb.modifiedtanimoto.**calc_mean_onbit_density**(*bitsets*, *number_of_bits*)

Calculate the mean density of bits that are on in bitsets collection.

Parameters

- **bitsets** (*list[pyroaring.BitMap]*) – List of fingerprints
- **number_of_bits** – Number of bits for all fingerprints

Returns Mean on bit density

Return type float

kripodb.modifiedtanimoto.**corrections**(*mean_onbit_density*)

Calculate corrections

See *similarity()* for explanation of corrections.

Parameters **mean_onbit_density** (*float*) – Mean on bit density

Returns S_T correction, S_{T0} correction

Return type float

kripodb.modifiedtanimoto.**similarities**(*bitsets1*, *bitsets2*, *number_of_bits*, *corr_st*, *corr_sto*, *cutoff*, *ignore_upper_triangle=False*)

Calculate modified tanimoto similarity between two collections of fingerprints

Excludes similarity of the same fingerprint.

Parameters

- **bitsets1** (*Dict{str, pyroaring.BitMap}*) – First dict of fingerprints with fingerprint label as key and pyroaring.BitMap as value
- **bitsets2** (*Dict{str, pyroaring.BitMap}*) – Second dict of fingerprints with fingerprint label as key and pyroaring.BitMap as value
- **number_of_bits** (*int*) – Number of bits for all fingerprints
- **corr_st** (*float*) – St correction
- **corr_sto** (*float*) – Sto correction
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **ignore_upper_triangle** (*Optional[bool]*) – When true returns similarity where label1 > label2, when false returns all similarities

Yields (fingerprint label 1, fingerprint label2, similarity score)

kripodb.modifiedtanimoto.**similarity**(*bitset1*, *bitset2*, *number_of_bits*, *corr_st*, *corr_sto*)

Calculate modified Tanimoto similarity between two fingerprints

Given two fingerprints of length n with a and b bits set in each fingerprint, respectively, and c bits set in both fingerprint, selected from a data set of fingerprint with a mean bit density of ρ_0, the modified Tanimoto similarity S_{MT} is calculated as

$$S_{MT} = (\frac{2-_0}{3})S_T + (\frac{1+_0}{3})S_{T0}$$

where ST is the standard Tanimoto coefficient

$$S_T = \frac{c}{a+b-c}$$

and Sr0 is the inverted Tanimoto coefficient

$$S_{T0} = \frac{n-a-b+c}{n-c}$$

Parameters

- **bitset1** (*pyroaring.BitMap*) – First fingerprint
- **bitset2** (*pyroaring.BitMap*) – Second fingerprint
- **number_of_bits** (*int*) – Number of bits for all fingerprints
- **corr_st** (*float*) – St correction
- **corr_sto** (*float*) – Sto correction

Returns modified Tanimoto similarity

Return type float

3.8 kripodb.pairs

Module handling generation and retrieval of similarity of fingerprint pairs

kripodb.pairs.**dump_pairs**(*bitsets1*, *bitsets2*, *out_format*, *out_file*, *out*, *number_of_bits*, *mean_onbit_density*, *cutoff*, *label2id*, *nomemory*, *ignore_upper_triangle=False*)

Dump pairs of bitset collection.

A pairs are rows of the bitset identifier of both bitsets with a similarity score.

Parameters

- **bitsets1** (*Dict{str, pyroaring.BitMap}*) – First dict of fingerprints with fingerprint label as key and pyroaring.BitMap as value
- **bitsets2** (*Dict{str, pyroaring.BitMap}*) – Second dict of fingerprints with fingerprint label as key and pyroaring.BitMap as value
- **out_format** – ‘tsv’ or ‘hdf5’
- **out_file** – Filename of output file where ‘hdf5’ format is written to.
- **out** (*File*) – File object where ‘tsv’ format is written to.
- **number_of_bits** (*int*) – Number of bits for all bitsets
- **mean_onbit_density** (*float*) – Mean on bit density

- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **label2id** – dict to translate label to id (string to int)
- **nomemory** – If true bitset2 is not loaded into memory
- **ignore_upper_triangle** – When true returns similarity where label1 > label2, when false returns all similarities

kripodb.pairs.**dump_pairs_hdf5**(*similarities_iter*, *label2id*, *expectedrows*, *out_file*)
Dump pairs in hdf5 file

Pro: * very small, 10 bytes for each pair + compression Con: * requires hdf5 library to access

Parameters

- **similarities_iter** (*Iterator*) – Iterator with tuple with fingerprint 1 label, fingerprint 2 label, similarity as members
- **label2id** (*dict*) – dict to translate label to id (string to int)
- **expectedrows** –
- **out_file** –

kripodb.pairs.**dump_pairs_tsv**(*similarities_iter*, *out*)
Dump pairs in tab delimited file

Pro: * when stored in sqlite can be used outside of Python Con: * big, unless output is compressed

Parameters

- **similarities_iter** (*Iterator*) – Iterator with tuple with fingerprint 1 label, fingerprint 2 label, similarity as members
- **out** (*File*) – Writeable file

kripodb.pairs.**merge**(*ins*, *out*)
Concatenate similarity matrix files into a single one.

Parameters

- **ins** (*list[str]*) – List of input similarity matrix filenames
- **out** (*str*) – Output similarity matrix filenames

Raises AssertionError – When nr of labels of input files is not the same

kripodb.pairs.**open_similarity_matrix**(*fn*)
Open read-only similarity matrix file.

Parameters **fn** (*str*) – Filename of similarity matrix

Returns A read-only similarity matrix object

Return type SimilarityMatrix | FrozenSimilarityMatrix

kripodb.pairs.**similar**(*query*, *similarity_matrix*, *cutoff*, *limit=None*)
Find similar fragments to query based on similarity matrix.

Parameters

- **query** (*str*) – Query fragment identifier
- **similarity_matrix** (*kripodb.db.SimilarityMatrix*) – Similarity matrix
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Yields *Tuple[(str, str, float)]* – List of (query fragment identifier, hit fragment identifier, similarity score) sorted on similarity score

kripodb.pairs.**similar_run**(*query*, *pairsdbfn*, *cutoff*, *out*)
Find similar fragments to query based on similarity matrix and write to tab delimited file.

Parameters

- **query** (*str*) – Query fragment identifier
- **pairsdbfn** (*str*) – Filename of similarity matrix file or url of kripodb webservice
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **out** (*File*) – File object to write output to

kripodb.pairs.**similarity2query**(*bitsets2*, *query*, *out*, *mean_onbit_density*, *cutoff*, *memory*)
Calculate similarity of query against all fingerprints in bitsets2 and write to tab delimited file.

Parameters

- **bitsets2** (kripodb.db.IntbitsetDict) –
- **query** (*str*) – Query identifier or beginning of it
- **out** (*File*) – File object to write output to
- **mean_onbit_density** (*flaot*) – Mean on bit density
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **memory** (*Optional[bool]*) – When true will load bitset2 into memory, when false it doesn't

kripodb.pairs.**total_number_of_pairs**(*fingerprint_filenames*)
Count number of pairs in similarity matrix files

Parameters **fingerprint_filenames** (*list[str]*) – List of file names of similarity matrices

Returns Total number of pairs

Return type int

3.9 kripodb.pharmacophores

kripodb.pharmacophores.**FEATURE_TYPES = [{'color': 'ff33cc', 'element': 'He', 'key': 'LI**
Types of pharmacophore feature types. List of dictionaries with the following keys –

- key, short identifier of type
- label, human readable label
- color, hex rrggbb color
- element, Element used in kripo pharmacophore sdfile for this type

class kripodb.pharmacophores.**PharmacophorePointsTable**(*h5file*, *expectedrows=0*)
Wrapper around pytables table to store pharmacohpore points

Parameters

- **h5file** (*tables.File*) – Pytables hdf5 file object which contains the pharmacophores table

- **expectedrows** (*int*) – Expected number of pharmacophores. Required when hdf5 file is created, helps optimize compression

Pharmacophore points of a fragment can be retrieved using:

```
points = table['frag_id1']
```

points is a list of points, each point is a tuple with following columns feature type key, x, y and z coordinate. The feature type key is defined in FEATURE_TYPES.

Number of pharmacophore points can be requested using:

```
nr_points = len(table)
```

To check whether fragment identifier is contained use:

```
'frag_id1' in table
```

add_dir(*startdir*)
Find *_pphore.sd.gz *_pphores.txt file pairs recursively in start directory and add them.

Parameters **startdir** (*str*) – Path to a start directory

read_phar(*infile*)
Read phar formatted file and add pharmacophore to self

Parameters **infile** – File object of phar formatted file

class kripodb.pharmacophores.**PharmacophoresDb**(*filename*, *mode='r'*, *expectedrows=0*, ***kwargs*)
Database for pharmacophores of fragments aka sub-pockets.

Parameters

- **filename** (*str*) – File name of hdf5 file to write or read pharmacophores to/from
- **mode** (*str*) – Can be 'r' for reading or 'w' for writing or 'a' for appending
- **expectedrows** (*int*) – Expected number of pharmacophores. Required when hdf5 file is created, helps optimize compression
- ****kwargs** – Passed to tables.open_file

Pharmacophore points of a fragment can be retrieved using:

```
points = db['frag_id1']
```

points is a list of points, each point is a tuple with following columns feature type key, x, y and z coordinate. The feature type key is defined in FEATURE_TYPES.

h5file
tables.File – Object representing an open hdf5 file

points
PharmacophorePointsTable – HDF5 table that contains pharmacophore points

add_dir(*startdir*)
Find *_pphore.sd.gz *_pphores.txt file pairs recursively in start directory and add them.

Parameters **startdir** (*str*) – Path to a start directory

append(*other*)
Append pharmacophores in other db to self

Parameters **other** (PharmacophoresDb) – The other pharmacophores database

close()
Closes the hdf5file

Instead of calling close() explicitly, use context manager:

```
with PharmacophoresDb('data/pharmacophores.h5') as db:
    points = db['frag_id1']
```

read_phar(*infile*)
Read phar formatted file and add pharmacophore to self

Parameters **infile** – File object of phar formatted file

write_phar(*outfile*, *frag_id=None*)
Write pharmacophore of frag_id as phar format to outfile

Parameters

- **outfile** (*file*) – File object to write to
- **frag_id** (*str*) – Fragment identifier, if None all pharmacophores are written

kripodb.pharmacophores.**as_phar**(*frag_id*, *points*)
Return pharmacophore in *.phar format.

See align-it for format description.

Parameters

- **frag_id** (*str*) – Fragment identifier
- **points** (*list*) – List of points where each point is (key,x,y,z)

Returns Pharmacophore is *.phar format

Return type str

kripodb.pharmacophores.**read_fragtxtfile**(*fragtxtfile*)
Read a fragment text file

Parameters **fragtxtfile** – Filename of fragment text file

Returns Dictionary where key is fragment identifier and value is a list of pharmacophore point indexes.

Return type dict

kripodb.pharmacophores.**read_fragtxtfile_as_file**(*fileobject*)
Read a fragment text file object which contains the pharmacophore point indexes for each fragment identifier.

File format is a fragment on each line, the line is space separated with fragment_identifier followed by the pharmacophore point indexes.

Parameters **fileobject** (*file*) – File object to read

Returns Dictionary where key is fragment identifier and value is a list of pharmacophore point indexes.

Return type dict

kripodb.pharmacophores.**read_pphore_gzipped_sdfile**(*sdfile*)
Read a gzipped sdfile which contains pharmacophore points as atoms

Parameters **sdfile** (*string*) – Path to filename

Returns List of Pharmacophore points

Return type list

kripodb.pharmacophores.**read_pphore_sdfile**(*sdfile*)
Read a sdfile which contains pharmacophore points as atoms

Parameters **sdfile** (*file*) – File object with sdfile contents

Returns List of pharmacophore points

Return type list

3.10 kripodb.pdb

class kripodb.pdb.**PdbReport**(*pdbids=None*, *fields=None*)
Client for the Custom Report Web Services of the RCSB PDB website

See http://www.rcsb.org/pdb/software/wsreport.do for more information.

Parameters

- **pdbids** (*List[str]*) – List of pdb identifiers to fetch. Default is ['*'] which fetches all.
- **fields** – (List[str]: List of fields to fetch. Default is ['structureTitle', 'compound', 'ecNo', 'uniprotAcc', 'uniprotRecommendedName'] See http://www.rcsb.org/pdb/results/reportField.do for possible fields.

url
str – Url of report, based on pdbids and fields.

fetch()
Fetch report from PDB website

Yields *dict* – Dictionary with keys same as ['structureId', 'chainID'] + self.fields

kripodb.pdb.**parse_csv_file**(*thefile*)
Parse csv file, yielding rows as dictionary.

The csv file should have an header.

Parameters **thefile** (*file*) – File like object

Yields *dict* – Dictionary with column header name as key and cell as value

3.11 kripodb.script

kripodb.script.**main**(*argv=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)
Main script function.

Calls run method of selected sub commandos.

Parameters **argv** (*list[str]*) – List of command line arguments

kripodb.script.**make_parser**()
Creates a parser with sub commands

Returns parser with sub commands

Return type argparse.ArgumentParser

kripodb.script.fragments.**make_fragments_parser**(*subparsers*)
Creates a parser for fragments sub commands

Parameters **subparsers** (*argparse.ArgumentParser*) – Parser to which to add sub commands to

kripodb.script.fingerprints.**make_fingerprints_parser**(*subparsers*)
Creates a parser for fingerprints sub commands

Parameters **subparsers** (*argparse.ArgumentParser*) – Parser to which to add sub commands to

kripodb.script.similarities.**make_similarities_parser**(*subparsers*)
Creates a parser for similarities sub commands

Parameters **subparsers** (*argparse.ArgumentParser*) – Parser to which to add sub commands to

kripodb.script.similarities.**read_fpneighpairs_file**(*inputfile*, *ignore_upper_triangle=False*)
Read fpneigh formatted similarity matrix file.

Parameters

- **inputfile** (*File*) – File object to read
- **ignore_upper_triangle** (*bool*) – Ignore upper triangle of input

Yields *Tuple((Str,Str,Float))* – List of (query fragment identifier, hit fragment identifier, similarity score)

kripodb.script.similarities.**simmatrix_export_run**(*simmatrixfn*, *outputfile*, *no_header*, *frag1*, *pdb*)
Export similarity matrix to tab delimited file

Parameters

- **simmatrixfn** (*str*) – (Compact) hdf5 similarity matrix filename
- **outputfile** (*file*) – Tab delimited output file
- **no_header** (*bool*) – Output no header
- **frag1** (*bool*) – Only output *frag1
- **pdb** (*str*) – Filename with pdb codes inside

kripodb.script.dive.**dense_dump_sc**(*sc*)
Dump dense matrix with zeros

3.12 kripodb.webservice

Module for Client for kripo web service

exception kripodb.webservice.client.**Incomplete**(*message*, *absent_identifiers*)

exception kripodb.webservice.client.**IncompleteFragments**(*absent_identifiers*, *fragments*)

exception kripodb.webservice.client.**IncompletePharmacophores**(*absent_identifiers*, *pharmacophores*)

class kripodb.webservice.client.**WebserviceClient**(*base_url*)
Client for kripo web service

Example

```
>>> client = WebserviceClient('http://localhost:8084/kripo')
>>> client.similar_fragments('3j7u_NDP_frag24', 0.85)
[{'query_frag_id': '3j7u_NDP_frag24', 'hit_frag_id': '3j7u_NDP_frag23', 'score':␣
↪0.8991}]
```

Parameters **base_url** (*str*) – Base url of web service. e.g. http://localhost:8084/kripo

fragments_by_id(*fragment_ids*, *chunk_size=100*)

Retrieve fragments by their identifier

Parameters

- **fragment_ids** (*List[str]*) – List of fragment identifiers
- **chunk_size** (*int*) – Number of fragment to retrieve in a single http request

Returns List of fragment information

Return type list[dict]

Raises *IncompleteFragments* – When one or more of the identifiers could not be found.

fragments_by_pdb_codes(*pdb_codes*, *chunk_size=450*)

Retrieve fragments by their PDB code

Parameters

- **pdb_codes** (*List[str]*) – List of PDB codes
- **chunk_size** (*int*) – Number of PDB codes to retrieve in a single http request

Returns List of fragment information

Return type list[dict]

Raises `requests.HTTPError` – When one of the PDB codes could not be found.

similar_fragments(*fragment_id*, *cutoff*, *limit=1000*)

Find similar fragments to query.

Parameters

- **fragment_id** (*str*) – Query fragment identifier
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Returns Query fragment identifier, hit fragment identifier and similarity score

Return type list[dict]

Raises `request.HTTPError` – When fragment_id could not be found

Kripo datafiles wrapped in a webservice

class kripodb.webservice.server.**KripodbJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

JSON encoder for KripoDB object types

Copied from http://flask.pocoo.org/snippets/119/

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    return JSONEncoder.default(self, o)
```

kripodb.webservice.server.**get_fragment_phar**(*fragment_id*)

Pharmacophore in phar format of fragment

Parameters **fragment_id** (*str*) – Fragment identifier

Returns Pharmacophore|problem

Return type flask.Response|connexion.lifecycle.ConnexionResponse

kripodb.webservice.server.**get_fragment_svg**(*fragment_id*, *width*, *height*)

2D drawing of fragment in SVG format

Parameters

- **fragment_id** (*str*) – Fragment identifier
- **width** (*int*) – Width of SVG in pixels
- **height** (*int*) – Height of SVG in pixels

Returns SVG document|problem

Return type flask.Response|connexion.lifecycle.ConnexionResponse

kripodb.webservice.server.**get_fragments**(*fragment_ids=None*, *pdb_codes=None*)

Retrieve fragments based on their identifier or PDB code.

Parameters

- **fragment_ids** (*List[str]*) – List of fragment identifiers
- **pdb_codes** (*List[str]*) – List of PDB codes

Returns List of fragment information

Return type list[dict]

Raises `werkzeug.exceptions.NotFound` – When one of the fragments_ids or pdb_code could not be found

kripodb.webservice.server.**get_similar_fragments**(*fragment_id*, *cutoff*, *limit*)

Find similar fragments to query.

Parameters

- **fragment_id** (*str*) – Query fragment identifier
- **cutoff** (*float*) – Cutoff, similarity scores below cutoff are discarded.
- **limit** (*int*) – Maximum number of hits. Default is None for no limit.

Returns List of dict with query fragment identifier, hit fragment identifier and similarity score

Return type list[dict]

Raises `werkzeug.exceptions.NotFound` – When the fragments_id could not be found

`kripodb.webservice.server.`**`get_version`**`()`

Returns Version of web service

Return type dict[version]

`kripodb.webservice.server.`**`serve_app`**(*similarities*, *fragments*, *pharmacophores*, *internal_port=8084*, *external_url='http://localhost:8084/kripo'*)

Serve webservice forever

Parameters

- **`similarities`** – Filename of similarity matrix hdf5 file
- **`fragments`** – Filename of fragments database file
- **`pharmacophores`** – Filename of pharmacophores hdf5 file
- **`internal_port`** – TCP port on which to listen
- **`external_url`** (*str*) – URL which should be used in Swagger spec

`kripodb.webservice.server.`**`wsgi_app`**(*similarities*, *fragments*, *pharmacophores*, *external_url='http://localhost:8084/kripo'*)

Create wsgi app

Parameters

- **`similarities`** (`SimilarityMatrix`) – Similarity matrix to use in webservice
- **`fragments`** (`FragmentsDb`) – Fragment database filename
- **`pharmacophores`** – Filename of pharmacophores hdf5 file
- **`external_url`** (*str*) – URL which should be used in Swagger spec

Returns connexion.App

CHAPTER 4

Command line interface

```
usage: kripodb [-h] [--version]
               {fingerprints,fragments,similarities,dive,serve,pharmacophores}
               ...
```

4.1 Positional Arguments

subcommand Possible choices: fingerprints, fragments, similarities, dive, serve, pharmacophores

4.2 Named Arguments

--version show program's version number and exit

4.3 Sub-commands:

4.3.1 fingerprints

Fingerprints

```
kripodb fingerprints [-h]
                     {import,export,meanbitdensity,similar,similarities,merge}
                     ...
```

Sub-commands:

import

Add Makebits file to fingerprints db

```
kripodb fingerprints import [-h] infile [infile ...] outfile
```

Positional Arguments

infile Name of makebits formatted fingerprint file (.tar.gz or not packed or - for stdin)

outfile Name of fingerprints db file

Default: "fingerprints.db"

export

Dump bitsets in fingerprints db to makebits file

```
kripodb fingerprints export [-h] infile outfile
```

Positional Arguments

infile Name of fingerprints db file

Default: "fingerprints.db"

outfile Name of makebits formatted fingerprint file (or - for stdout)

meanbitdensity

Compute mean bit density of fingerprints

```
kripodb fingerprints meanbitdensity [-h] [--out OUT] fingerprintsdb
```

Positional Arguments

fingerprintsdb Name of fingerprints db file (default: "fingerprints.db")

Default: "fingerprints.db"

Named Arguments

--out Output file, default is stdout (default: -)

Default: -

similar

Find the fragments closests to query based on fingerprints

```
kripodb fingerprints similar [-h] [--mean_onbit_density MEAN_ONBIT_DENSITY]
                             [--cutoff CUTOFF] [--memory]
                             fingerprintsdb query out
```

Positional Arguments

fingerprintsdb	Name of fingerprints db file
	Default: "fingerprints.db"
query	Query identifier or beginning of it
out	Output file tabdelimited (query, hit, score)

Named Arguments

--mean_onbit_density	Mean on bit density (default: 0.01)
	Default: 0.01
--cutoff	Set Tanimoto cutoff (default: 0.55)
	Default: 0.55
--memory	Store bitsets in memory (default: False)
	Default: False

similarities

Output formats: * tsv, tab separated id1,id2, similarity * hdf5, hdf5 file constructed with pytables with a, b and score, but but a and b have been replaced

by numbers and similarity has been converted to scaled int

When input has been split into chunks, use *–ignore_upper_triangle* flag for computing similarities between same chunk. This prevents storing pair a->b also as b->a.

```
kripodb fingerprints similarities [-h] [--out_format {tsv,hdf5}]
                                  [--fragmentsdbfn FRAGMENTSDBFN]
                                  [--mean_onbit_density MEAN_ONBIT_DENSITY]
                                  [--cutoff CUTOFF] [--nomemory]
                                  [--ignore_upper_triangle]
                                  fingerprintsfn1 fingerprintsfn2 out_file
```

Positional Arguments

fingerprintsfn1	Name of reference fingerprints db file
fingerprintsfn2	Name of query fingerprints db file
out_file	Name of output file (use - for stdout)

Named Arguments

--out_format Possible choices: tsv, hdf5

Format of output (default: "hdf5")

Default: "hdf5"

--fragmentsdbfn Name of fragments db file (only required for hdf5 format)

--mean_onbit_density Mean on bit density (default: 0.01)

Default: 0.01

--cutoff Set Tanimoto cutoff (default: 0.45)

Default: 0.45

--nomemory Do not store query fingerprints in memory (default: False)

Default: False

--ignore_upper_triangle Ignore upper triangle (default: False)

Default: False

merge

Combine fingerprints databases into a single new one

```
kripodb fingerprints merge [-h] ins [ins ...] out
```

Positional Arguments

ins Input fingerprints database files

out Output fingerprints database file

4.3.2 fragments

Fragments

```
kripodb fragments [-h] {shelve,sdf,pdb,filter,merge,export_sd} ...
```

Sub-commands:

shelve

Add fragments from shelve to sqlite

```
kripodb fragments shelve [-h] [--skipdups] shelvefn fragmentsdb
```

Positional Arguments

shelvefn

fragmentsdb Name of fragments db file (default: “fragments.db”)

Default: “fragments.db”

Named Arguments

--skipdups Skip duplicates, instead of dieing one first duplicate

Default: False

sdf

Add fragments sdf to sqlite

```
kripodb fragments sdf [-h] sdffns [sdffns ...] fragmentsdb
```

Positional Arguments

sdffns SDF filename

fragmentsdb Name of fragments db file (default: “fragments.db”)

Default: “fragments.db”

pdb

Add pdb metadata from RCSB PDB website to fragment sqlite db

```
kripodb fragments pdb [-h] fragmentsdb
```

Positional Arguments

fragmentsdb Name of fragments db file (default: “fragments.db”)

Default: “fragments.db”

filter

Filter fragments database

```
kripodb fragments filter [-h] [--pdbs PDBS] [--matrix MATRIX] input output
```

Positional Arguments

input	Name of fragments db input file
output	Name of fragments db output file, will overwrite file if it exists

Named Arguments

--pdbs	Keep fragments from any of the supplied pdb codes, one pdb code per line, use - for stdin
--matrix	Keep fragments which are in similarity matrix file

merge

Combine fragments databases into a single new one

```
kripodb fragments merge [-h] ins [ins ...] out
```

Positional Arguments

ins	Input fragments database files
out	Output fragments database file

export_sd

Export molblocks of all fragments as SDF file

```
kripodb fragments export_sd [-h] fragmentsdb sdfile
```

Positional Arguments

fragmentsdb	Input fragments database file
sdfile	Output SDF file

4.3.3 similarities

Similarity matrix

```
kripodb similarities [-h]
                     {similar,merge,export,import,filter,freeze,thaw,fpneigh2tsv,
↪histogram}
                     ...
```

Sub-commands:

similar

Find the fragments closets to query based on similarity matrix

```
kripodb similarities similar [-h] [--out OUT] [--cutoff CUTOFF]
                             pairsdbfn query
```

Positional Arguments

pairsdbfn	hdf5 similarity matrix file or base url of kripodb webservice
query	Query fragment identifier

Named Arguments

--out	Output file tab delimited (query, hit, similarity score)
	Default: -
--cutoff	Similarity cutoff (default: 0.55)
	Default: 0.55

merge

Combine pairs files into a new file

```
kripodb similarities merge [-h] ins [ins ...] out
```

Positional Arguments

ins	Input pair file in hdf5_compact format
out	Output pair file in hdf5_compact format

export

Export similarity matrix to tab delimited file

```
kripodb similarities export [-h] [--no_header] [--frag1] [--pdb PDB]
                            simmatrixfn outputfile
```

Positional Arguments

simmatrixfn	Compact hdf5 similarity matrix filename
outputfile	Tab delimited output file, use - for stdout

Named Arguments

--no_header Output no header (default: False)

Default: False

--frag1 Only output *frag1 fragments (default: False)

Default: False

--pdb Only output fragments which are from pdb code in file, one pdb code per line (default: None)

import

When input has been split into chunks, use *–ignore_upper_triangle* flag for similarities between same chunk. This prevents storing pair a->b also as b->a.

```
kripodb similarities import [-h] [--inputformat {tsv,fpneigh}]
                            [--nrrows NRROWS] [--ignore_upper_triangle]
                            inputfile fragmentsdb simmatrixfn
```

Positional Arguments

inputfile Input file, use - for stdin

fragmentsdb Name of fragments db file (default: “fragments.db”)

Default: “fragments.db”

simmatrixfn Compact hdf5 similarity matrix file, will overwrite file if it exists

Named Arguments

--inputformat Possible choices: tsv, fpneigh

tab delimited (tsv) or fpneigh formatted input (default: “fpneigh”)

Default: “fpneigh”

--nrrows Number of rows in inputfile (default: 65536)

Default: 65536

--ignore_upper_triangle Ignore upper triangle (default: False)

Default: False

filter

Filter similarity matrix

```
kripodb similarities filter [-h] [--fragmentsdb FRAGMENTSDB | --skip SKIP]
                            input output
```

Positional Arguments

input	Input hdf5 similarity matrix file
output	Output hdf5 similarity matrix file, will overwrite file if it exists

Named Arguments

--fragmentsdb	Name of fragments db file, fragments in it will be kept as well as their pair counter parts.
--skip	File with fragment identifiers on each line to skip

freeze

Optimize similarity matrix for reading

```
kripodb similarities freeze [-h] [-f FRAME_SIZE] [-m MEMORY] [-l LIMIT] [-s]
                            in_fn out_fn
```

Positional Arguments

in_fn	Input pairs file
out_fn	Output array file, file is overwritten

Named Arguments

-f, --frame_size	Size of frame (default: 100000000)
	Default: 100000000
-m, --memory	Memory cache in Gigabytes (default: 1)
	Default: 1
-l, --limit	Number of pairs to copy, None for no limit (default: None)
-s, --single_sided	Store half matrix (default: False)
	Default: False

thaw

Optimize similarity matrix for writing

```
kripodb similarities thaw [-h] [--nonzero_fraction NONZERO_FRACTION]
                          in_fn out_fn
```

Positional Arguments

in_fn	Input packed frozen matrix file
out_fn	Output pairs file, file is overwritten

Named Arguments

--nonzero_fraction	Fraction of pairs which have score above threshold (default: 0.012) Default: 0.012

fpneigh2tsv

Convert fpneigh formatted file to tab delimited file

```
kripodb similarities fpneigh2tsv [-h] inputfile outputfile
```

Positional Arguments

inputfile	Input file, use - for stdin
outputfile	Tab delimited output file, use - for stdout

histogram

Distribution of similarity scores

```
kripodb similarities histogram [-h] [-f FRAME_SIZE] [-r] [-l]
                               inputfile outputfile
```

Positional Arguments

inputfile	Filename of similarity matrix hdf5 file
outputfile	Tab delimited output file, use - for stdout

Named Arguments

-f, --frame_size	Size of frame (default: 100000000) Default: 100000000
-r, --raw_score	Return raw score (16 bit integer) instead of fraction score Default: False
-l, --lower_triangle	Return scores from lower triangle else return scores from upper triangle Default: False

4.3.4 dive

DiVE visualization utils

```
kripodb dive [-h] {fragments,dump,export} ...
```

Sub-commands:

fragments

Export fragments as DiVE formatted sphere

```
kripodb dive fragments [-h] [--onlyfrag1] inputfile outputfile
```

Positional Arguments

inputfile	Name of fragments db input file
outputfile	Name of fragments dive output file, use - for stdout

Named Arguments

--onlyfrag1	Only *_frag1 (default: False) Default: False

dump

Dump dense matrix with zeros

```
kripodb dive dump [-h] [--frag1only] inputfile outputfile
```

Positional Arguments

inputfile	Name of dense similarity matrix
outputfile	Name of output file, use - for stdout

Named Arguments

--frag1only	Only *frag1 (default: False) Default: False

export

Writes props for DiVE visualization

```
kripodb dive export [-h] [--propnames PROPNAMES] [--props PROPS]
                    [--pdbtags PDBTAGS]
                    fragmentsdb uniprot_annot
```

Positional Arguments

fragmentsdb Name of fragments db input file

uniprot_annot **Uniprot download accession 2 gene symbol, family mapping.** Fetch "http://www.uniprot.org/uniprot/?query=database:pdb&format=tab&columns=id,genes(PREFERRED),families,database(PDB)"

Named Arguments

--propnames Name of prop names file

Default: kripo.propnames.txt

--props Name of props file

Default: kripo.props.txt

--pdbtags Tag pdb in file by filename

4.3.5 serve

Serve similarity matrix, fragments db and pharmacophores db as webservice

```
kripodb serve [-h] [--internal_port INTERNAL_PORT]
              [--external_url EXTERNAL_URL]
              similarities fragments pharmacophores
```

Positional Arguments

similarities Filename of similarity matrix hdf5 file

fragments Filename of fragments sqlite database file

pharmacophores Filename of pharmacophores hdf5 file

Named Arguments

--internal_port TCP port on which to listen (default: 8084)

Default: 8084

--external_url URL which should be used in Swagger spec (default: "http://localhost:8084/kripo")

Default: "http://localhost:8084/kripo"

4.3.6 pharmacophores

Pharmacophores

```
kripodb pharmacophores [-h] {add,get,filter,merge,import,sd2phar} ...
```

Sub-commands:

add

Add pharmacophores from directory to database

```
kripodb pharmacophores add [-h] [--nrrows NRROWS] startdir pharmacophoresdb
```

Positional Arguments

startdir Directory to start finding *.pphores.sd.gz and *.pphores.txt files in

pharmacophoresdb Name of pharmacophore db file

Named Arguments

--nrrows **Number of expected pharmacophores,** only used when database is created (default: 65536)

Default: 65536

get

Retrieve pharmacophore of a fragment

```
kripodb pharmacophores get [-h] [--query QUERY] [--output OUTPUT]
                           pharmacophoresdb
```

Positional Arguments

pharmacophoresdb Name of pharmacophore db file

Named Arguments

--query Query fragment identifier

--output Phar formatted text file

Default: -

filter

Filter pharmacophores

```
kripodb pharmacophores filter [-h] [--fragmentsdb FRAGMENTSDB]
                              inputfn outputfn
```

Positional Arguments

inputfn	Name of input pharmacophore db file
outputfn	Name of output pharmacophore db file

Named Arguments

--fragmentsdb	Name of fragments db file, fragments present in db are passed (default: "fragments.db") Default: "fragments.db"

merge

Merge pharmacophore database files into new one

```
kripodb pharmacophores merge [-h] ins [ins ...] out
```

Positional Arguments

ins	Input pharmacophore database files
out	Output pharmacophore database file

import

Convert phar formatted file to pharmacophore database file

```
kripodb pharmacophores import [-h] [--nrrows NRROWS] infile outfile
```

Positional Arguments

infile	Input phar formatted file
outfile	Output pharmacophore database file

Named Arguments

--nrrows **Number of expected pharmacophores,** only used when database is created (default: 65536)

Default: 65536

sd2phar

Convert sd formatted pharmacophore file to phar formatted file

```
kripodb pharmacophores sd2phar [-h] [--frag_id FRAG_ID] infile outfile
```

Positional Arguments

infile Input sd formatted file

outfile Output phar formatted file

Named Arguments

--frag_id Fragment identifier

Default: “frag”

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

G

H

I

K

L

M

N

O

P