CanGraph

Release 0.9

Pablo Marcos

Aug 30, 2022

CONTENTS

CHAPTER

ONE

TO-DO LIST

The following problems are known issues which will be solved, but are yet to be adressed:

Todo: CAMBIAR NOMBRE A LOS MESH PARA INDICAR EL TIPO. AÑADIR NAME A LOS WIKIDATA

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 3.)

Todo: FIX THE REPEAT TRANSACTION FUNCTION

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 4.)

Todo: Match partial InChIs based on DICE-MACCS

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 5.)

Todo: QUE FUNCIONE -> ACTUALMENTE ESTA SECCION RALENTIZA MAZO

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 6.)

Todo: CHECK APOC IS INSTALLED

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 7.)

Todo: FIX MAIN

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 8.)

Todo: MERGE BY INCHI, METANETX ID

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 9.)

Todo: Fix find_protein_interactions_in_metanetx

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 10.)

Todo: Mover esa funcion de setup a misc

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 11.)

Todo: EDIT conf.py

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 12.)

Todo: Detail Schema Changes

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 13.)

Todo: Document the following Schema Changes: For Subject, we have a composite PK: Exposome_Explorer_ID, Age, Gender e Information Now, more diseases will have a WikiData_ID and a related MeSH. This will help with networking. And, this diseases dont even need to be a part of a cancer! The Gene nodes no longer exist in the full db? -> They do

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/main.py:docstring of CanGraph.main.main, line 15.)

Todo: Make it download the image

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/miscelaneous.py:doc of CanGraph.miscelaneous.call_db_schema_visualization, line 6.)

Todo: What if `interactive=False`?

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/setup.py:docstring of CanGraph.setup.setup_drugbank, line 7.)

Todo: What if `interactive=False`?

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/setup.py:docstring of CanGraph.setup.setup_exposome, line 7.)

Todo: In some other parts of the script, sequences are being added as properties on Protein nodes. A common format should be set.

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifyDrugBank/b of CanGraph.GraphifyDrugBank.build_database.add_sequences, line 12.)

Todo: Investigate https://stackoverflow.com/questions/14026217/using-neo4j-distinct-and-order-by-on-different-properties

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifyDrugBank/b of CanGraph.GraphifyDrugBank.build_database.add_targets_enzymes_carriers_and_transporters, line 26.)

Todo: It would be nice to be able to distinguish between experimental and predicted properties

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifyHMDB/buil of CanGraph.GraphifyHMDB.build_database.add_biological_properties, line 16.)

Todo: It would be nice to be able to distinguish between experimental and predicted properties

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifyHMDB/buil of CanGraph.GraphifyHMDB.build_database.add_experimental_properties, line 16.)

Todo: It would be nice to be able to distinguish between experimental and predicted properties

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifyHMDB/buil of CanGraph.GraphifyHMDB.build_database.add_predicted_properties, line 16.)

Todo: This file is really big. It could be divided into smaller ones.

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifySMPDB/bui of CanGraph.GraphifySMPDB.build_database.add_pathways, line 12.)

Todo: Why is the SMPDB_ID property called like that and not SMDB_ID?

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/GraphifySMPDB/bui of CanGraph.GraphifySMPDB.build_database.add_proteins, line 15.)

Todo: Might include P684 “Orthologues” for more info (it crashed java)

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/QueryWikidata/build of CanGraph.QueryWikidata.build_database.add_gene_info, line 13.)

Todo: Might include P527 “has part or parts” for more info (it crashed java)

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/QueryWikidata/build
of CanGraph.QueryWikidata.build_database.add_metabolite_info, line 16.)

Todo: ADD ROLE to metabolite interactions

(The original entry is located in /home/pablo/Documentos/Trabajos del cole/UNI/Master/Prácticas/Work/CanGraph/QueryWikidata/build
of CanGraph.QueryWikidata.build_database.add_more_drug_info, line 13.)

CHAPTER

TWO

CANGRAPH PACKAGE

This Git Project, created as part of my Master's Intenship at IARC, contains a series of scripts that pulls information from a series of **five** databases from their native format (XML, CSV, etc) into a common, GraphML format, using a shared schema that has been defined to minimize the number of repeated nodes and properties. This databases are:

- Exposome-Explorer: A hand-curated, high-quality database of associations between metabolites, food intakes and outakes and different diseases, specially cancers.
- Human Metabolome DataBase: An detailed, electronic database containing detailed information about small molecule metabolites found in the human body.
- DrugBank: A unique bioinformatics and cheminformatics resource that combines detailed drug data with comprehensive drug target information.
- Small Molecule Pathway Database: An interactive database containing more than 618 small molecule pathways found in humans, More than 70% of which are unique to this DB
- WikiData: The world's largest collaboratively generated collection of Open Data worldwide.

Each of then have their unique advantages and disadvantages (size, quality, etc) but they have been chosen to work together and help in identifying metabolites and their potential cancer associations at IARC.

With regards to the schema, it can be consulted in detail in the `new-schema.graphml` file, which can itself be opened in Neo4J by calling: `CALL apoc.import.graphml("new-schema.graphml", {useTypes:true, storeNodeIds:false, readLabels:True})` after placing it in your Neo4J's import directory (you can find it in the settings shown after starting the server with `sudo neo4j start`). It consists of a simplification of all the nodes present on the `old-schema.graphml` file (which itself represents the five different schemas that our five databases natively presented), arrived at by merging nodes and changing relationship names so that they are unique (and, thus, more actionable). One property, `LabelName` has been added as a dummy name to generate the image you can see in the header.

This repo contains two kind of scripts: first, some `build_database.py` scripts, which contain the information to re-build the databases in the common format from scratch, and are located in subsequent subfolders named after the database they come from (more info can be consulted on them on their respective READMEs) and a common `main.py` script, which can be used to query for sub-networks based solely on info presented on a **sample_input.csv** database of identified compounds which we would like to annotate.

2.1 Intallation

To use this script, you should first clone it into your personal computer. The easiest way to do this is to git clone the repo:

1. Install git (if not already installed). On linux: `sudo apt install git`
2. Clone the repo: `git clone https://codeberg.org/FlyingFlamingo/graphify-databases`
3. Step into the directory `cd graphify-databases`

Once the project has been installed, you **must** run `setup.py`, a preparation script that guides you through the process of installing all five databases on your computer, so that then we can correctly process them and generate the sub-networks. You should also install the required python modules and run the setup script:

4. PIP install all dependencies: `pip install -r requirements.txt`
5. Run the setup script: `python3 setup.py`

Once this has been done, you are ready to start using the main script!

NOTE: If you do not wish to use git, you can manually download the repo by clicking here

2.2 Usage

To generate this sub-networks (the original idea of the project) you should run:

```
python3 main.py neo4jadress databaseusername databasepassword databasefolder inputfile
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **databasename**: the name of the database in use. If using the free version, there will only be one database per project (neo4j being the default name); if using the pro version, you can specify an alternate name here
- **databasepassword**: the passowrd for the **databasename** DataBase. Since the arguments are passed by BaSH onto python3, you might need to escape special characters
- **databasefolder**: The folder indicated to `setup.py` as the one where your databases will be stored
- **inputfile**: The location of the CSV file in which the program will search for metabolites. This file should be a Comma-Separated file, with the following format: `MonoisotopicMass, SMILES, InChIKey, Name, InChI, Identifier, ChEBI`

2.3 Subpackages

2.3.1 CanGraph.ExposomeExplorer package

This package, created as part of my Master's Intenship at IARC, transitions the exposome-explorer database (a high quality, hand-curated database containing associations of foods and chemical compounds with cancer) to Neo4J format in an automated way, providing an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 main.py neo4jadress databasename databasepassword csvfolder
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **databasename**: the name of the database in use. If using the free version, there will only be one database per project (neo4j being the default name); if using the pro version, you can specify an alternate name here
- **databasepassword**: the passowrd for the **databasename** DataBase. Since the arguments are passed by BaSH onto python3, you might need to escape special characters
- **csvfolder**: The folder where the CSV files for the Exposome Explorer database are stored. This CSVs have to be manually exported from the (confidential) database itself, and are **NOT** equivalent to those found in exposome-explorer download's page

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

The package consists of the following modules:

CanGraph.ExposomeExplorer.build_database module

A python module that provides the necessary functions to transition the Exposome Explorer database to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.ExposomeExplorer.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_auto_units(*tx*, *filename*)

Shows the correlations between two units, converted using the rubygem 'https://github.com/masa16/phys-units' which standarizes units of measurement for our data

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_cancer_associations(*tx*, *filename*)

Imports the 'cancer_associations' database as a relation between a given Cancer and a Measurement

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_cancers(*tx*, *filename*)

Adds "Cancer" nodes from Exposome-Explorer's cancers.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_cohorts(*tx*, *filename*)

Adds "Cohort" nodes from Exposome-Explorer's cohorts.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_components(*tx*, *filename*)

Adds "Metabolite" nodes from Exposome-Explorer's components.csv This is because this components are, in fact, metabolites, either from food or from human metabolism

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_correlations(*tx*, *filename*)

Imports the 'correlations' database as a relation between to measurements: the intake_id, a food taken by the organism and registered using dietary questionnaires and the excretion_id, a chemical found in human biological samples, such that, when one takes one component, one will excrete the other. Data comes from epidemiological studies where dietary questionnaires are administered, and biomarkers are measured in specimens

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_experimental_methods(*tx*, *filename*)

Adds "ExperimentalMethod" nodes from Exposome-Explorer's experimental_methods.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_measurements(*tx*, *filename*)

Adds "Measurement" nodes from Exposome-Explorer's measurements.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_measurements_stuff(*tx*, *filename*)

A massive and slow-running function that creates ALL the relations between the 'measurements' table and all other related tables:

- units: The units in which a given measurement is expressed
- components: The component which is being measured
- samples: The sample from which a measurement is taken
- experimental_methods: The method used to take a measurement

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_metabolomic_associations(*tx*, *filename*)

Imports the 'metabolomic_associations' database as a relation between to measurements: the intake_id, a food taken by the organism and registered using dietary questionnaires and the excretion_id, a chemical found in human biological samples, such that, when one takes one component, one will excrete the other. Data comes from Metabolomics studies seeking to identify putative dietary biomarkers.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_microbial_metabolite_identifications(*tx*, *filename*)

Imports the relations pertaining to the "microbial_metabolite_identifications" table. A component (i.e. a metabolite) can be identified as a Microbial Metabolite, which means it has an equivalent in the microbiome. This can have a given reference and a tissue (BioSpecimen) in which it occurs.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_microbial_metabolite_info`(*tx*, *filename*)

Adds "Metabolite" nodes from Exposome-Explorer's microbial_metabolite_identifications.csv These represent all metabolites that have been re-identified as present, for instance, in the microbiome. To distinguish this metabolites from the only-human "Components", a Microbial_Metabolite = "True" label has been added

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_publications`(*tx*, *filename*)

Adds "Publication" nodes from Exposome-Explorer's publications.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_reproducibilities`(*tx*, *filename*)

Adds "Reproducibility" nodes from Exposome-Explorer's reproducibilities.csv These represent the conditions under which a given study/measurement was carried

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_samples`(*tx*, *filename*)

Adds "Sample" nodes from Exposome-Explorer's samples.csv From a Sample, one can take a series of measurements

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_specimens(*tx*, *filename*)

Adds "BioSpecimen" nodes from Exposome-Explorer's specimens.csv A biospecimen is a type of tissue where a measurement can originate, such as orine, csf fluid, etc

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_subjects(*tx*, *filename*)

Adds "Subject" nodes from Exposome-Explorer's subjects.csv

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_units(*tx*, *filename*)

Adds "Unit" nodes from Exposome-Explorer's units.csv A unit can be converted into other (for example, for normalization)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

build_from_file(*databasepath*, *Neo4JImportPath*, *driver*, *keep_counts_and_displayeds=True*, *keep_cross_properties=False*)

A function able to build a portion of the Exposome-Explorer database in graph format, provided that at least one "Component" (Metabolite) node is present in said database. It works by using that node as an starting point from which to search in the rest of the Exposome_Explorer database, finding related nodes there.

Parameters

- **databasepath** (*str*) – The path to the database where all Exposome-Explorer CSVs are stored
- **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data
- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use

- **keep_counts_and_displayeds** (*bool*) – Whether to keep the properties ending with `_count` & `displayed_` that, although present in the original DB, might be considered not useful for us.
- **keep_cross_properties** (*bool*) – Whether to keep the properties used to cross-reference in the original Neo4J database.

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

import_csv(*tx*, *filename*, *label*)

Imports a given CSV into Neo4J. This CSV **must** be present in Neo4J's Import Path

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported
- **label** (*str*) – The label of the Neo4J nodes that will be imported, with the columns of the CSV being its properties.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For this to work, you HAVE TO have APOC availaible on your Neo4J installation

---

remove_counts_and_displayeds(*inputfile*, *outputfile*)

Removes `_count` & `displayed_` text-strings from a given file, so that, when processing it with the other functions present in this document, they ignore the columns containing said text-strings, which represent properties which are considered not useful for our program. This is. of course, not the most elegant, but it works.

Parameters

- **inputfile** (*str*) – The path to the file from which `_count` & `displayed_ text-strings are to be removed
- **outputfile** (*str*) – The path of the file where the contents of the replaced file will be written.

Returns The function does not have a return; instead, it transforms ``inputfile`` into `outputfile`

remove_cross_properties(*tx*)

Removes some properties that were added by the other functions present in this script, that are used to cross-reference the different tables in the Relational Database EE comes from, and that, in a Graph Database, are no longer necessary.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

CanGraph.ExposomeExplorer.main module

A python module that leverages the functions present in the *build_database* module to recreate the exposome-explorer database using a graph format and Neo4J, and then provides an GraphML export file.

Please note that, to work, the functions here pre-suppose you have access to Exposome-Explorer internal CSVs, and that you have placed them under a folder provided as `sys.argv[4]`. These CSVs are confidential, and can only be accessed under request to the *International Agency for Research on Cancer*.

For more details on how to run this script, please consult the package's README

main(*args*)

The function that executes the code

2.3.2 CanGraph.GraphifyDrugBank package

This package, created as part of my Master's Intenship at IARC, imports nodes from the DrugBank Database (a high quality database containing a drugs and proteins with some characteristics) to Neo4J format in an automated way, providing an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 main.py neo4jadress username databasepassword filename
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **username**: the username for your neo4j instance. Remember, the default is neo4j
- **password**: the passowrd for your database. Since the arguments are passed by BaSH onto python3, you might need to escape special characters
- **filename**: the database's file name. For practical purpouses, it **must** be present in ./xmlfolder

Please note that there are two kinds of functions in the associated code: those that use python f-strings, which themselves contain text that *cannot* be directly copied into Neo4J (for instance, double brackets have to be turned into simple brackets) and normal multi-line strings, which can. This is because f-strings allow for variable customization, while normal strings dont.

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

Important Notices

- To access the Drugbank database, you have to previously request access at: https://go.drugbank.com/public_users/sign_up
- Some XML tags have been intentionally not processed; for example, the tag didn't seem to dd anything new, and the and tags are likely not relevant for our project. The tag has no info (check `cat full\ database.xml | grep "<ahfs-codes>"`) and the tag seemed like too much work (same for). nOT USE because not of use to us
- Some :Proteins might also be :Drugs and viceversa: this makes the schema difficult to understand, but provides more meaningful data. This is also why some relations (`RELATED_WITH`, for instance) dont have a relation: this way, we avoid duplicates.

The package consists of the following modules:

CanGraph.GraphifyDrugBank.build_database module

A python module that provides the necessary functions to transition the DrugBank database to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.GraphifyDrugBank.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_atc_codes(*tx*, *filename*)

Creates "ATC" nodes based on XML files obtained from the DrugBank website. These represent the different ATC codes a Drug can be related with (including an small taxonomy)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_categories(*tx*, *filename*)

Creates "Category" nodes based on XML files obtained from the DrugBank website. These represent the different MeSH IDs a Drug can be related with

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Each category seems to have an associated MeSH ID. Maybe could rename nodes as MeSH?

add_dosages(*tx*, *filename*)

Creates "Dosage" nodes based on XML files obtained from the DrugBank website. These represent the different Dosages that a Drug should be administered at.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: Using CREATE might generate duplicate nodes, but there was no unique characteristic to MERGE nodes into.

add_drug_interactions(*tx*, *filename*)

Creates `(d)-[r:RELATED_WITH_DRUG]-(dd)` interactions between "Drug" nodes, whether they existed before or not. These are intentionally non-directional, as they should be related with each other.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_drugs(*tx*, *filename*)

Creates "Drug" nodes based on XML files obtained from the DrugBank website, adding some essential identifiers and external properties.

See also:

This way of working has been taken from William Lyon's Blog

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Since Publications dont have any standard identificator, they are created using the "Title"

---

add_experimental_properties(*tx*, *filename*)

Adds some experimental properties to existing "Drug" nodes based on XML files obtained from the DrugBank website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_external_equivalents(*tx*, *filename*)

Adds some external equivalents to existing "Drug" nodes based on XML files obtained from the DrugBank website. This should be "exact matches" of the Drug in other databases.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: The main reason to add them as "External-Equivalents" is because I felt these IDs where of not much use (and are thus easier to eliminate due to their common label)

`add_external_identifiers`(*tx*, *filename*)

Adds some external identifiers to existing "Drug" nodes based on XML files obtained from the DrugBank website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: These also adds a "Protein" label to any "Drug"-labeled nodes which have a "UniProtKB"-ID among their properties. NOTE that this can look confusing in the DB Schema!!!

`add_general_references`(*tx*, *filename*)

Creates "Publication" nodes based on XML files obtained from the DrugBank website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Since not all nodes present a "PubMed_ID" field (which would be ideal to uniquely-identify Publications, as the "Text" field is way more prone to typos/errors), nodes will be created using the "Authors" field. This means some duplicates might exist, which should be accounted for.

`add_manufacturers`(*tx*, *filename*)

Creates "Company" nodes based on XML files obtained from the DrugBank website. These represent the different Companies that manufacture a Drug's compound (not just package it)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_mixtures(*tx*, *filename*)

Creates "Mixture" nodes based on XML files obtained from the DrugBank website. These are the mixtures of existing Drugs, which may or may not be on the market.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: This doesn't seem of much use, but has been added nonetheless just in case.

---

add_packagers(*tx*, *filename*)

Creates "Company" nodes based on XML files obtained from the DrugBank website. These represent the different Companies that package a Drug's compounds (not the ones that manufacture them)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_pathways_and_relations(*tx*, *filename*)

Adds "Pathway" nodes based on XML files obtained from the DrugBank website. It also adds some relations between Drugs and Proteins (which, remember, could even be the same kind of node) It is also able to tag both a Protein's and a Drug¡s relation with a given Pathway In general, a Pathway involves a collection of Enzymes, Drugs and Proteins, with a SMPDB_ID (cool for interconnexion!)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: This function uses a "double UNWIND" clause, which means that we are only representing <pathways> tags with <enzymes> tags inside. Fortunately, this seems to seldom not happen, so it should represent no problem.

add_products(*tx*, *filename*)

Creates "Product" nodes based on XML files obtained from the DrugBank website. These are the individual medicaments that have been approved (or not) by the FDA

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: Using CREATE means that duplicates will appear; unfortunately, I couldnt any unique_id field to use as ID when MERGEing the nodes. This should be accounted for.

`add_sequences`(*tx*, *filename*)

Creates "Sequence" nodes based on XML files obtained from the DrugBank website. These represent the AminoAcid sequence of Drugs that are of a peptidic nature.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Todo: In some other parts of the script, sequences are being added as properties on Protein nodes. A common format should be set.

`add_targets_enzymes_carriers_and_transporters`(*tx*, *filename*, *tag_name*)

A *REALLY HUGE* function. It takes a filename and a tag_name, and gets info and creates "Protein" nodes with tag_name set as their role. It also adds a bunch of additional info, such as Publications, Targets, Actions, GO_IDs, PFAMs and/or some External IDs

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported
- **tag_name** (*str*) – The type of Protein node you want to import; it must be one of ["enzymes", "carriers", "transporters"] It is recommended that you run this function thrice, once for each type of protein

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: We are using a bunch of concatenated UNWINDs, which force the existance of all elements in the UNWIND chain. This might remove some elements, but this is a HUUUUUGE database, and, to be honest, most things seem to almost always be present. An example is References and Polypeptides; Since there seem to be more References that Polypeptides, we try to UNWIND those first. The same can be said on the rest of UNWINDS: as we have external-id >>>>>>>>>> go-classifier >>>> pfam >> synonyms (in order of *occurrence*. not *number* of tags) , we UNWIND in that order to mitigate data loss

Note: To fix repetitions in properties such as Actions or Synonyms (caused by the HUGE number of UNWINDs), we tried lots of different strategies, finally coming up with SET p.Synonyms = replace(p.Synonyms, synonym._text + ",", ""). This is cool! But means there will always be a trailing comma (removing it was not easy in this same transaction, though it could (TODO?) be done at the end.

Note: <tag_name>

Todo: Investigate https://stackoverflow.com/questions/14026217/using-neo4j-distinct-and-order-by-on-different-properties

`add_taxonomy`(*tx*, *filename*)

Creates "Taxonomy" nodes based on XML files obtained from the DrugBank website. These represent the "kind" of Drug we are dealing with (Family, etc)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: It only creates relationships in the Kingdom -> Super Class -> Class -> Subclass direction, and from any node -> Drug. This means that, if any member of the Kingdom -> Super Class -> Class -> Subclass is absent, the line will be broken; hopefully in that case a new Drug will come in to rescue and settle the relation!

Warning: Some nodes without labels might be created if names are null: This has to be accounted for later on in the process

`build_from_file`(*newfile*, *driver*)

A function able to build a portion of the DrugBank database in graph format, provided that one XML is supplied to it. This can either be the `full_database.xml` file that you can get in DrugBank's website, or a splitted version of it, with just one item per file (which is recommended due to memory limitations)

Parameters

- **newfile** (*str*) – The path of the XML file to import
- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

CanGraph.GraphifyDrugBank.main module

A python module that leverages the functions present in the *build_database* module to recreate the DrugBank database using a graph format and Neo4J, and then provides an GraphML export file.

Please note that, to work, the functions here pre-suppose you have a DrugBank account with access to the whole Database. You have to prevously apply for it at: https://go.drugbank.com/public_users/sign_up, and place the `full_database.xml` file at `./xmlfolder`.

For more details on how to run this script, please consult the package's README

main()

The function that executes the code

2.3.3 CanGraph.GraphifyHMDB package

This script, created as part of my Master's Intenship at IARC, imports nodes from the Human Metabolome Database (a high quality, database containing a list of metabolites and proteins associated to different diseases) to Neo4J format in an automated way, providing an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 main.py neo4jadress username databasepassword
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **username**: the username for your neo4j instance. Remember, the default is neo4j
- **password**: the passowrd for your database. Since the arguments are passed by BaSH onto python3, you might need to escape special characters

Please note that there are two kinds of functions in the associated code: those that use python f-strings, which themselves contain text that *cannot* be directly copied into Neo4J (for instance, double brackets have to be turned into simple brackets) and normal multi-line strings, which can. This is because f-strings allow for variable customization, while normal strings dont.

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

Important Notices

- Please ensure you have internet access, enough espace in your hard drive (around 5 GB) and read-write access in `./xmlfolder`. The files needed to build the database will be stored there.
- There are two kinds of high-level nodes stored in this database: "Metabolites", which are individual compounds present in the Human Metabolome; and "Proteins", which are normally enzimes and are related to one or multiple metabolites. There are different types of metabolites, but they were all imported in the same way; their origin can be differenced by the "" field on the corresponding "Concentration" nodes. You could run a query such as: `MATCH (n:Metabolite)-[r:MEASURED_AT]-(c:Concentration) RETURN DISTINCT c.Biospecimen`
- Some XML tags have been intentionally not processed; for example, the tag seemed like too much info unrelated to our project, or the tags, which could be useful but seemed to only link to external DBs

The package consists of the following modules:

CanGraph.GraphifyHMDB.build_database module

A python module that provides the necessary functions to transition the HMDB database to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.GraphifyHMDB.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_biological_properties(*tx*, *filename*)

Adds biological properties to existing "Metabolite" nodes based on XML files obtained from the HMDB website. In this case, only properties labeled as <predicted_properties> are added.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Another option would have been to auto-add all the properties, and name them using RETURN "Predicted " + apoc.text.capitalizeAll(replace(kind, "_", " ")), value; however, this way we can select and not duplicate / overwrite values.

Todo: It would be nice to be able to distinguish between experimental and predicted properties

---

add_concentrations_abnormal(*tx*, *filename*)

Creates "Concentration" nodes based on XML files obtained from the HMDB website. In this function, only metabolites that are labeled as "abnormal_concentration" are added.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Here, an UNWIND clause is used instead of a FOREACH clause. This provides better performance, since, unlike FOREACH, UNWIND does not process rows with empty values

---

Warning: Using the CREATE row forces the creation of a Concentration node, even when some values might be missing. However, this means some bogus nodes could be added, which MUST be accounted for at the end of the DB-Creation process.

add_concentrations_normal(*tx*, *filename*)

Creates "Concentration" nodes based on XML files obtained from the HMDB website. In this function, only metabolites that are labeled as "normal_concentration" are added.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Here, an UNWIND clause is used instead of a FOREACH clause. This provides better performance, since, unlike FOREACH, UNWIND does not process rows with empty values

---

Warning: Using the CREATE row forces the creation of a Concentration node, even when some values might be missing. However, this means some bogus nodes could be added, which MUST be accounted for at the end of the DB-Creation process.

add_diseases(*tx*, *filename*)

Creates "Publication" nodes based on XML files obtained from the HMDB website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Here, an UNWIND clause is used instead of a FOREACH clause. This provides better performance, since, unlike FOREACH, UNWIND does not process rows with empty values (and, logically, there should be no Publication if there is no Disease)

Note: Publications are created with a (m)-[r:CITED_IN]->(p) relation with Metabolite nodes. If one wants to find the Publication nodes related to a given Metabolite/Disease relation, one can use:

```
MATCH p=()-[r:RELATED_WITH]->()
  WITH split(r.PubMed_ID, ",") as pubmed
    UNWIND pubmed as find_this
    MATCH (p:Publication)
      WHERE p.PubMed_ID = find_this
RETURN p
```

---

add_experimental_properties(*tx*, *filename*)

Adds properties to existing "Metabolite" nodes based on XML files obtained from the HMDB website. In this case, only properties labeled as <experimental_properties> are added.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Another option would have been to auto-add all the properties, and name them using RETURN "Experimental " + apoc.text.capitalizeAll(replace(kind, "_", " ")), value; however, this way we can select and not duplicate / overwrite values.

Todo: It would be nice to be able to distinguish between experimental and predicted properties

---

`add_gene_properties`(*tx*, *filename*)

Adds some properties to existing "Protein" nodes based on XML files obtained from the HMDB website. In this case, properties will mostly relate to the gene from which the protein originates.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: We are not creating "Gene" nodes (even though each protein comes from a given gene) because we believe not enough information is being given about them.

`add_general_references`(*tx*, *filename*, *type_of*)

Creates "Publication" nodes based on XML files obtained from the HMDB website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Since not all nodes present a "PubMed_ID" field (which would be ideal to uniquely-identify Publications, as the "Text" field is way more prone to typos/errors), nodes will be created using the "Authors" field. This means some duplicates might exist, which should be accounted for.

Note: Unlike the rest, here we are not matching metabolites, but ALSO proteins. This is intentional.

---

add_go_classifications(*tx*, *filename*)

Creates "Gene Ontology" nodes based on XML files obtained from the HMDB website. This relates each protein to some GO-Terms

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_metabolite_associations(*tx*, *filename*)

Adds associations contained in the "protein" file, between proteins and metabolites.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Like he "add_metabolite_associations" function, this creates non-directional relationships (m)-[r:ASSOCIATED_WITH]-(p) ; this helps duplicates be detected.

Note: The "ON CREATE SET" clause for the "Name" param ensures no overwriting

---

add_metabolite_references(*tx*, *filename*)

Creates references for relations betweens Protein nodes and Metabolite nodes

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: Unfortunately, Neo4J makes it really, really, really difficult to work with XML, and so, this time, a r.PubMed_ID list with the references could not be created. Nonetheless, I considered adding this useful.

add_metabolites(*tx*, *filename*)

Creates "Metabolite" nodes based on XML files obtained from the HMDB website, adding some essential identifiers and external properties.

See also:

This way of working has been taken from William Lyon's Blog

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_predicted_properties`(*tx*, *filename*)

Adds properties to existing "Metabolite" nodes based on XML files obtained from the HMDB website. In this case, only properties labeled as <predicted_properties> are added.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Another option would have been to auto-add all the properties, and name them using RETURN "Predicted " + apoc.text.capitalizeAll(replace(kind, "_", " ")), value; however, this way we can select and not duplicate / overwrite values.

Todo: It would be nice to be able to distinguish between experimental and predicted properties

---

`add_protein_associations`(*tx*, *filename*)

Creates "Protein" nodes based on XML files obtained from the HMDB website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Unlike the "add_protein" function, this creates Proteins based on info on the "Metabolite" files, not on the "Protein" files themselves. This could mean node duplication, but, hopefully, the MERGE by Accession will mean that this duplicates will be catched.

`add_protein_properties`(*tx*, *filename*)

Adds some properties to existing "Protein" nodes based on XML files obtained from the HMDB website. In this case, properties will mostly relate to the protein itself.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: The "signal_regions" and the "transmembrane_regions" properties were left out because, after a preliminary search, they were mostly empty

---

add_proteins(*tx*, *filename*)

Creates "Protein" nodes based on XML files obtained from the HMDB website.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: We are not creating "Gene" nodes (even though each protein comes from a given gene) because we believe not enough information is being given about them.

---

add_taxonomy(*tx*, *filename*)

Creates "Taxonomy" nodes based on XML files obtained from the HMDB website. These represent the "kind" of metabolite we are dealing with (Family, etc)

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the XML file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: It only creates relationships in the Kingdom -> Super Class -> Class -> Subclass direction, and from any node -> Metabolite. This means that, if any member of the Kingdom -> Super Class -> Class -> Subclass is absent, the line will be broken; hopefully in that case a new metabolite will come in to rescue and settle the relation!

---

build_from_metabolite_file(*newfile*, *driver*)

A function able to build a portion of the HMDB database in graph format, provided that one "Metabolite" XML is supplied to it. This are downloaded separately from the website, as all the files that are not `hmdb_proteins.zip`, and can be presented either as the full file, or as a splitted version of it, with just one item per file (which is recommended due to memory limitations)

Parameters

- **newfile** (*str*) – The path of the XML file to import
- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

build_from_protein_file(*newfile*, *driver*)

A function able to build a portion of the HMDB database in graph format, provided that one "Protein" XML is supplied to it. This are downloaded separately from the website, as `hmdb_proteins.zip`, and can be presented either as the full file, or as a splitted version of it, with just one item per file (which is recommended due to memory limitations)

Parameters

- **newfile** (*str*) – The path of the XML file to import
- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

CanGraph.GraphifyHMDB.main module

A python module that leverages the functions present in the *build_database* module to recreate the HMDB database using a graph forma and Neo4J, and then provides an GraphML export file.

Please note that, to work, the functions here pre-suppose you have internet access, which will be used to download HMDB's XMLs under `./xmlfolder/` (please ensure you have read-write access there).

For more details on how to run this script, please consult the package's README

main()

The function that executes the code

2.3.4 CanGraph.GraphifySMPDB package

This script, created as part of my Master's Intenship at IARC, transitions the Small Molecule Pathway Database (a high quality database containing associations betweem metabolites, proteins and metabolomic pathways) to Neo4J format in an automated way, providing an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 main.py neo4jadress databasename databasepassword
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **databasename**: the name of the database in use. If using the free version, there will only be one database per project (neo4j being the default name); if using the pro version, you can specify an alternate name here
- **databasepassword**: the passowrd for the **databasename** DataBase. Since the arguments are passed by BaSH onto python3, you might need to escape special characters

NOTE: The files will be downloaded to ./csvfolder, so please run the script somewhere you have read/write permissions

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

Important Notices

- Please ensure you have internet access, enough espace in your hard drive (around 5 GB) and read-write access in `./csvfolder`. The files needed to build the database will be stored there.
- Since the "Structure" files at smpdb.ca seemed to be super complicated to import, we decided against doing so. However, regarding the future, they shouldn't be overlooked, as they might include useful info
- The "PW ID" column from the "Pathways" table, and the "Pathway Subject" from the "Metabolite" tables may correlate (both start with PW). However, they seem to use different formats, so we have decided against including them in the Final Database

---

The package consists of the following modules:

CanGraph.GraphifySMPDB.build_database module

A python module that provides the necessary functions to transition the SMPDB database to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.GraphifySMPDB.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_metabolites(*tx*, *filename*)

Adds "Metabolite" nodes to the database, according to individual CSVs present in the SMPDB website

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Some of the node's properties might be set to "null" (important in order to work with it)

Note: This database clearly differentiates Metabolites and Proteins, so no overlap is accounted for

---

add_pathways(*tx*, *filename*)

Adds "Pathways" nodes to the database, according to individual CSVs present in the SMPDB website Since this is done after the creation of said pathways in the last step, this will most likely just annotate them.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Todo: This file is really big. It could be divided into smaller ones.

---

add_proteins(*tx*, *filename*)

Adds "Protein" nodes to the database, according to individual CSVs present in the SMPDB website

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Some of the node's properties might be set to "null" (important in order to work with it)

Note: This database clearly differentiates Metabolites and Proteins, so no overlap is accounted for

---

Todo: Why is the SMPDB_ID property called like that and not SMDB_ID?

Warning: Since no unique identifier was found, CREATE had to be used (instead of merge). This might create duplicates. which should be accounted for.

add_sequence(*tx*, *seq_id*, *seq_name*, *seq_type*, *seq*, *seq_format='FASTA'*)

Adds "Pathways" nodes to the database, according to the sequences presented in FASTA files from the SMPDB website

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **seq_id** (*str*) – The UniProt Database Identifier for the sequence that is been imported
- **seq_name** (*str*) – The Name (i.e. FASTA header) of the Sequence that is been imported
- **seq_type** (*bool*) – The type of the sequence; can be either of ["DNA", "PROT"]
- **seq** (*str*) – The seuqnce that is been imported; a text chain of nucleotides or aminoacids, identified by their acronyms
- **seq_format** (*str*) – The format the sequence is provided under; default is "FASTA", but its optional

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

build_from_file(*databasepath*, *filepath*, *Neo4JImportPath*, *driver*, *filetype*)

A function able to build a portion of the SMPDB in graph format, provided that one CSV is supplied to it. This CSVs are downloaded from the website, and can be presented either as the full file, or as a splitted version of it, with just one item per file (which is recommended due to memory limitations)

Since file title represents a different pathway, the function automatically picks up and import the relative pathway node.

Parameters

- **databasepath** (*str*) – The path to the database where all SMPDB CSVs are stored
- **filepath** (*str*) – The path to the current file being imported
- **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data
- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use
- **filetype** (*bool*) – The type of file being imported; one of ether ["Metabolite", "Protein"]- If the file is a FASTA sequence store, this will be auto-detected.

Returns

This function modifies the Neo4J Database as desired, but does not produce any particular return.

Note: Since this adds a ton of low-resolution nodes, maybe have this db run first?

CanGraph.GraphifySMPDB.main module

A python module that leverages the functions present in the *build_database* module to recreate the SMPDB database using a graph format and Neo4J, and then provides an GraphML export file.

Please note that, to work, the functions here pre-suppose you have internet access, which will be used to download HMDB's CSVs under `./csvfolder/` (please ensure you have read-write access there).

For more details on how to run this script, please consult the package's README

import_genomic_seqs()

Imports the `smpdb_gene.fasta` file from the SMPDB Database. The function assumes the file is availaible at `./csvfolder/smpdb_gene.fasta`

Parameters **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

import_metabolites(*filename*, *Neo4JImportPath*)

Imports "Metabolite" files from the SMPDB Database. The function assumes `filename` is availaible at `./csvfolder/smpdb_metabolites/`

Parameters

- **filename** (*str*) – The name of the CSV file that is being imported
- **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

import_pathways(*Neo4JImportPath*)

Imports the `smpdb_pathways.csv` file from the SMPDB Database. The function assumes the file is availaible at `./csvfolder/smpdb_gene.fasta`

Parameters **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

import_proteic_seqs()

Imports the `smpdb_protein.fasta` file from the SMPDB Database. The function assumes the file is availaible at `./csvfolder/smpdb_protein.fasta`

Parameters **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

import_proteins(*filename*, *Neo4JImportPath*)

Imports "Protein" files from the SMPDB Database. The function assumes `filename` is availaible at `./csvfolder/smpdb_proteins/`

Parameters

- **filename** (*str*) – The name of the CSV file that is being imported
- **Neo4JImportPath** (*str*) – The path from which Neo4J is importing data

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

main()

The function that executes the code

2.3.5 CanGraph.MeSHandMetaNetX package

This script, created as part of my Master's Intenship at IARC, transitions the MetaNetX DataBase (a high quality database that reconciles nomenclature for metabolites, proteins and metabolomic pathways) to Neo4J format in an automated way, providing an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 main.py neo4jadress databasename databasepassword tsvfolder
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **databasename**: the name of the database in use. If using the free version, there will only be one database per project (neo4j being the default name); if using the pro version, you can specify an alternate name here
- **databasepassword**: the passowrd for the **databasename** DataBase. Since the arguments are passed by BaSH onto python3, you might need to escape special characters
- **tsvfolder**: The folder where the TSV files for the MetaNetX DataBase are to be downloaded. This will be done automatically by the script.

The script will download the MetaNetX files, re-build the database using a graph format, and then annotate existing nodes with their KEGG Pathway IDs and MeSH IDs, using some web services.

NOTE: Since the files will be downloaded to `csvfolder`, please select somewhere you have read/write permissions

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

Important Notices

- Please ensure you have internet access, enough espace in your hard drive (around 5 GB) and read-write access in `tsvfolder`. The files needed to build the database will be stored there.
- Part of the content is taken from TSVs downloaded from MetaNetX's download page, and some of the data is acessed through its SPARQL service. This, perhaps surprising, design choice, was taken due to three things: first, MetaNetX does not provide peptXref files, so these necessarily *must come* from the RDF service, if we want to replicate the whole database; second, not all data could be taken from SPARQL due to potential timeouts, so at least some of the data *must come* from the TSVs; and, third, some of the functions where already built for the general script, so it made sense to reuse them here :p

---

The package consists of the following modules:

CanGraph.MeSHandMetaNetX.build_database module

A python module that provides the necessary functions to transition the MetaNetX database (and related MeSH terms and KEGG IDs) to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.MeSHandMetaNetX.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_chem_isom(*tx*, *filename*)

A CYPHER query that loads the *chem_isom.tsv* file availaible at the MetaNetX site, using a graph format.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For performance, it is recommended to split the file in 1 subfile for each row in the DataBase

---

add_chem_prop(*tx*, *filename*)

A CYPHER query that loads the *chem_prop.tsv* file availaible at the MetaNetX site, using a graph format.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For performance, it is recommended to split the file in 1 subfile for each row in the DataBase

---

add_chem_xref(*tx*, *filename*)

A CYPHER query that loads the *chem_xref.tsv* file availaible at the MetaNetX site, using a graph format.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For performance, it is recommended to split the file in 1 subfile for each row in the DataBase

add_comp_prop(*tx*, *filename*)

A CYPHER query that loads the *comp_prop.tsv* file availaible at the MetaNetX site, using a graph format.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For performance, it is recommended to split the file in 1 subfile for each row in the DataBase

add_comp_xref(*tx*, *filename*)

A CYPHER query that loads the *comp_xref.tsv* file availaible at the MetaNetX site, using a graph format.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **filename** (*str*) – The name of the CSV file that is being imported

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: For performance, it is recommended to split the file in 1 subfile for each row in the DataBase

Note: Some identifiers present the CL/cl prefix. Since I could not find what this prefix refers to, and since it only pertains to one single MetaNetX ID, we did not take them into account

Note: The "description" field in the DataBase is ignored, since it seems to be quite similar, but less useful, than the "name" field from comp_prop, which is more coherent with our pre-existing schema

add_mesh_by_name()

A function that adds some MeSH nodes to any existing nodes, based on their Name property.

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: neo4j.Session.run

Return type str

Note: Only exact matches work here, which is not ideal.

add_pept()

A CYPHER query that all the protein availaible at the MetaNetX site, using a graph format and SPARQL.

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: neo4j.Session.run

Return type str

Note: SPARQL was only used here because, unlike with the other files, there is no download available; also, given there are few proteins, Neo4J is able to process it without running out of memory (unlike what happened with the other fields)

Note: This is an **autocommit transaction**. This means that, in order to not keep data in memory (and make running it with a huge amount of data) more efficient, you will need to add `:auto ` when calling it from the Neo4J browser, or call it as `session.run( clean_database() )` from the driver.

add_prefixes()

Add some prefixes necessary for all MetaNetX queries to work. This are kept together since adding extra prefixes does not increase computation time

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: neo4j.Session.run

Return type str

build_from_file(*filename*, *driver*)

A function able to build a portion of the MetaNetX database in graph format, provided that one MetaNetX CSV is supplied to it. This CSVs are downloaded from the website, and can be presented either as the full file, or as a splitted version of it, with just one item per file (which is recommended due to memory limitations). If you want all the database to be imported, you should run this function with all the CSVs that form it, as portrayed in the *main* module

Parameters

- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use
- **filename** (*str*) – The name of the CSV file that is being imported

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

find_protein_data_in_metanetx()

A SPARQL function that annotates Protein nodes in an exiting Neo4J database by using the information provided by MetaNetX

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

Note: This function is partly a duplicate of *self.find_protein_interactions_in_metanetx()*, which was split to prevent timeouts

---

`find_protein_interactions_in_metanetx()`

A SPARQL function that finds the Metabolites a given Protein (based on its UniProt_ID) interacts with, using MetaNetX.

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

Note: We are not using peptXref: since all proteins in MetaNetX come from UniProt, there is no use here

---

`get_identifiers(from_sparql=False)`

Part of a CYPHER query that processes the outcome from a SPARQL query that searches for information on MetaNetX It takes an original metabolite (n) and a row variable, which should have columns named external_identifier, cross_refference, InChIKey, InChI, SMILES, Formula and Mass with the adequated format; it is basically a code-reuser, not intended to be used separately.

Parameters `from_sparql` (*bool*) – A True/False param defining whether the identifiers are being parsed from a SPARQL query; default is False (i.e. imported from file)

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

Note: All HMDB matches might create a Metabolite without CHEBI_ID or CAS_Number, which would violate our schema. This will be later on accounted for.

Note: Some keys, such as VMH_ID, are not merged into their own node, but rather added to an existing one. This is because this do not prevously exist in our Schema, and might be changed in the future.

Note: We dont care about overwriting InChI and InChIKey because they are necessarily unique; the same is true for Mass and Formula, as they are not all that important. However, for HMDB ID and others, we will take care not to overwrite, which could mess up the DB

`get_kegg_pathways_for_metabolites()`

A function that finds the Pathways a given Metabolite (based on its Kegg_ID) is a part of, using KEGG. This uses genome.jp's dbget web service, since I honestly could not find a way to use KEGG's SPARQL service (https://www.genome.jp/linkdb/linkdb_rdf.html) for that.

See also:

Another possibility could be using Kegg's Rest API

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

`read_synonyms_in_metanetx`(*tx*, *querytype=None*, *query=None*)

A SPARQL function that finds synonyms for metabolites, proteins or drugs based on a given *query*, using MetaNetX. At the same time, it is able to annotate them a bit, adding Name, InChI, InChIKey, SMILES, Formula, Mass, some External IDs, and finding whether the metabolite in question has any known isomers, anootating if so.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **querytype** (*str*) – The type of query that is being searched for. One of ["Name","KEGG_ID","ChEBI_ID","HMDB_ID","InChI","InChIKey"]; default is `None` (i.e. the function will not process anything).
- **query** (*str*) – The query we are searching for; must be of type `querytype`

Note: This is intended to be run as a read_transaction, only returning synonyms present in the DB. No modifications will be applied.

`write_synonyms_in_metanetx`(*query=None*)

A SPARQL function that finds synonyms for metabolites, proteins or drugs in an existing Neo4J database, using MetaNetX. At the same time, it is able to annotate them a bit, adding Name, InChI, InChIKey, SMILES, Formula, Mass, some External IDs, and finding whether the metabolite in question has any known isomers, anootating if so.

Parameters **query** (*str*) – The type of query that is being searched for. One of ["Name","KEGG_ID","ChEBI_ID","HMDB_ID","InChI","InChIKey"]; default is None (i.e. the function will not process anything).

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

Note: This is intended to be run as a write_transaction, modifying the existing database.

CanGraph.MeSHandMetaNetX.main module

A python module that leverages the functions present in the *build_database* module to recreate the MetaNetX database using a graph format and Neo4J, and then provides an GraphML export file. It also annotates related MeSH_IDs and KEGG Pathway IDs

Please note that, to work, the functions here pre-suppose you have internet access, which will be used to download MetaNetX's TSVs under a folder provided as `sys.argv[4]`. (please ensure you have read-write access there) and query some web SPARQL and REST web services.

For more details on how to run this script, please consult the package's README

`main`()

The function that executes the code

2.3.6 CanGraph.QueryWikidata package

This script, created as part of my Master's Intenship at IARC, imports nodes from the WikiData SPARQL Service, creating a high-quality representation of the data therein. Although wikidata is manually curated using the Wiki principles, some publications have found it might be a good source of information for life sciences, specially due to the breadth of information it contains. It also provides an export in GraphML format.

To run, it uses `alive_progress` to generate an interactive progress bar (that shows the script is still running through its most time-consuming parts) and the `neo4j` python driver. This requirements can be installed using: `pip install -r requirements.txt`.

To run the script itself, use:

```
python3 build_database.py neo4jadress username databasepassword
```

where:

- **neo4jadress**: is the URL of the database, in neo4j:// or bolt:// format
- **username**: the username for your neo4j instance. Remember, the default is neo4j
- **password**: the passowrd for your database. Since the arguments are passed by BaSH onto python3, you might need to escape special characters

Please note that there are two kinds of functions in the associated code: those that use python f-strings, which themselves contain text that *cannot* be directly copied into Neo4J (for instance, double brackets have to be turned into simple brackets) and normal multi-line strings, which can. This is because f-strings allow for variable customization, while normal strings dont.

An archived version of this repository that takes into account the gitignored files can be created using: `git archive HEAD -o ${PWD##*/}.zip`

Finally, please node that the general philosophy and approach of the queries have been taken from Towards Data Science, a genuinely useful web site.

Important Notices

- Please ensure you have internet access, which will be used to connect to Wikidata's SPAQL endpoint and gather the necessary info.
- As Neo4J can run out of "Java Heap Space" if the number of nodes/properties to add is too high, the script has been divided in order to minimize said number: for instance, only nodes with a `wikidata_id` ending in a given number from 0 to 9 are processed at a time. This does not decrease performance, since these nodes would have been processed nontheless, but makes the script more reliable.
- What does impact performance, however, is having different functions for adding cancers, drugs, metabolites, etc, instead of having just one match for each created cancer node. This makes WikiData have to process more queries that are less heavy, which makes it less likely to time-out, but causes the script to run more slowly.
- The Neo4J server presents a somewhat unstable connection that is sometimes difficult to keep alive, as it tends to be killed by the system when you so much as look at it wrong. To prevent this from happening, you are encouraged to assign a high-priority to the server's process by using the `nice` or `renice` commands in Linux (note that the process will be called "Java", not "Neo4J")
- Another measure taken to prevent Neo4J's unreliability from stopping the script is the `misc.repeat_transaction` function, which insists a given number of times until either the problem is fixed or the error persists. This is because Neo4J tends to: random disconnects, run out of java heap space, explode... and WikiData tends to give server errors, have downtimes during the **14+ hours** the script takes to run, etc.
- The data present in the "graph.graphml" file comes from WikiData, and was provided by this service free of charge and of royalties under the permissive CC-0 license.

The package consists of the following modules:

CanGraph.QueryWikidata.build_database module

A python module that provides the necessary functions to transition selected parts of the Wikidata database to graph format, either from scratch importing all the nodes (as showcased in *CanGraph.QueryWikidata.main*) or in a case-by-case basis, to annotate existing metabolites (as showcased in *CanGraph.main*).

add_cancer_info(*tx*, *number=None*)

Adds info to "Cancer" nodes for which its WikiData_ID ends in a given number. This way, only some of the nodes are targeted, and the Java Machine does not run out of memory

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **number** (*int*) – From 0 to 9, the number under which the WikiData_IDs to process should ends. This allows us tho divide the work, although its not very elegant.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_causes(*tx*, *number=None*)

Creates drug nodes related with each of the "Cancer" nodes already on the database

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **number** (*int*) – From 0 to 9, the number under which the WikiData_IDs to process should ends. This allows us tho divide the work, although its not very elegant.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_drug_external_ids(*tx*, *query='Wikidata'*)

Adds some external IDs to any "Drug" nodes already present on the database. Since the PDB information had too much values which caused triple duplicates that overcharged the system, they were intentionally left out.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **query** (*str*) – One of ["DrugBank_ID","WikiData_ID"], a way to identify the nodes for which external IDs will be added.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_drugs(*tx*, *number=None*)

Creates drug nodes related with each of the "Cancer" nodes already on the database

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

- **number** (*int*) – From 0 to 9, the number under which the WikiData_IDs to process should ends. This allows us tho divide the work, although its not very elegant.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_gene_info(*tx*)

A Cypher Query that adds some external IDs and properties to "Gene" nodes already existing on the database. This query forces the genes to have a "found_in_taxon:homo_sapiens" label. This means that any non-human genes will not be annotated (.. TODO:: delete those)

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Genomic Start and ends keep just the 2nd position, as reported in wikidata

---

Todo: Might include P684 "Orthologues" for more info (it crashed java)

---

add_genes(*tx*, *number=None*)

Creates gene nodes related with each of the "Cancer" nodes already on the database

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **number** (*int*) – From 0 to 9, the number under which the WikiData_IDs to process should ends. This allows us tho divide the work, although its not very elegant.

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

add_metabolite_info(*tx*, *query='ChEBI_ID'*)

A Cypher Query that adds some external IDs and properties to "Metabolite" nodes already existing on the database. Two kind of metabolites exist: those that are encoded by a given gene, and those that interact with a given drug. Both are adressed here, since they are similar, and, most likely, instances of proteins.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **query** (*str*) – One of ["DrugBank_ID","WikiData_ID"], a way to identify the nodes for which external IDs will be added; default is "WikiData_ID"

Returns

A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

- This function forces all metabolites to have a "found_in_taxon:human" target
- The metabolites are not forced to be proteins, but if they are, this is kept in the "instance_of" record

Return type neo4j.work.result.Result

Todo: Might include P527 "has part or parts" for more info (it crashed java)

`add_more_drug_info`(*tx*, *query='WikiData_ID'*)

Creates some nodes that are related with each of the "Drug" nodes already existing on the database: routes of administration, targeted metabolites and approved drugs that tehy are been used in

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **query** (*str*) – One of ["DrugBank_ID","WikiData_ID"], a way to identify the nodes for which external IDs will be added; default is "WikiData_ID"

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Todo: ADD ROLE to metabolite interactions

Note: This transaction has been separated in order to keep response times low

---

`add_toomuch_metabolite_info`(*tx*)

A function that adds loads of info to existing "Metabolite" nodes. This was left out, first because it might be too much information, (specially when it is already availaible by clicking the "url" field), and because, due to it been so much, it crashes the JVM.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`add_wikidata_to_mesh`(*tx*)

A function that adds some MeSH nodes and WikiData_IDs to existing nodes, based on their name. Since WikiData is not really specified on this, this wont add much info

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

`find_instance_of_cancer`(*tx*)

A Neo4J Cypher Statment that queries wikidata for instances of "Cancer" nodes already present on the Database. Since this are expected to only affect humans, this subclasses should also, only affect humans

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

find_subclass_of_cancer(*tx*)

A Neo4J Cypher Statment that queries wikidata for subclasses of "Cancer" nodes already present on the Database. Since this are expected to only affect humans, this subclasses should also, only affect humans

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

initial_cancer_discovery(*tx*)

A Neo4J Cypher Statment that queries wikidata for Human Cancers. Since using the "afflicts:human" tag didnt have much use here, I used a simple workaround: Query wikidata for all humans, and, among them, find all of this for which their cause of death was a subclass of "Cancer" (Q12078). Unfortunaltely, some of them were diagnosed "Cancer" (Q12078), which is too general, so I removed it.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

CanGraph.QueryWikidata.main module

A python module that leverages the functions present in the *build_database* module to recreate selected parts of the the Wikidata database using a graph format and Neo4J, and then provides an GraphML export file.

Please note that, to work, the functions here pre-suppose you have internet access, which will be used to access Wikidata's SPAQL endpoint and write info to the Neo4J database

For more details on how to run this script, please consult the package's README

main()

The function that executes the code

2.4 CanGraph.deploy module

A python module that simplifies deploying the different formats the program can present itself as:

- A web documentation that is made using sphinx
- A PDF manual likewise made, also using LaTeX
- A git repo where the program can be accessed and version-tagged
- And, in the future… a singularity container!

2.4.1 CanGraph.deploy Usage

To use this module:

```
usage: python3 deploy.py [-h] [-m] [-d] [-w] [-p]
```

Named Arguments

-m, --main	deploy code to your dev branch
-d, --dev	deploy code to your main branch
-w, --web	deploys the documentation to the 'pages' web site, depending on which is activated
-p, --pdf	generates a PDF version of the sphinx manual, and saves it to the repo

Note: For this program to work, the Git environment **has to be set up first**. You can ensure this by using: `CanGraph.setup.setup_git`

2.4.2 CanGraph.deploy Functions

This module is comprised of:

args_parser()

Parses the command line arguments into a more usable form, providing help and more

Returns A dictionary of the different possible options for the program as keys, specifying their set value. If no command-line arguments are provided, the help message is shown and the program exits.

Return type argparse.ArgumentParser

Note: Note that, in Google Docstrings, if you want a multi-line `Returns` comment, you have to start it in a different line :(

Note: The return **must** be of type `argparse.ArgumentParser` for the `argparse` directive to work and autogen docs

deploy_code(*branch='dev'*)

Deploys code from a given branch to the corresponding remote.

Parameters **branch** (*str*) – The name of the branch of the **local** git repo that we want to deploy

Note: Normally, the `pages` branch should be published using *deploy_webdocs*, which in theory would be weird to publish without updating the docs first

deploy_pdf_manual(*docs_folder='./docs/'*, *work_dir='.'*, *manual_location='./CanGraph_Manual.pdf'*, *prechecks_done=False*, *custom_domain=None*)

Parses the command line arguments into a more usable form, providing help and more

Generates the PDF docs guide, and publishes it in `manual_location`

Parameters

- **docs_folder** (*str*) – The path to sphinx's docs folder, where the tests will be run; by default `./docs/`
- **work_dir** (*str*) – The current Working Directory; by default
- **prechecks_done** (*bool*) – Whether the prechecks present in *~CanGraph.deploy.make_sphinx_prechecks* have already been made
- **custom_domain** (*str*) – A custom domain to deploy de docs to.
- **manual_location** (*str*) – The location (**including filename**) of the finalised PDF manual, relative to the location of the script

Returns Whether the prechecks have already been done; always True if the function is run

Return type bool

deploy_webdocs(*docs_folder='./docs/'*, *work_dir='.'*, *prechecks_done=False*, *custom_domain=None*)

Generates the HTML web docs, and publishes it both to Github and Codeberg pages

Parameters

- **docs_folder** (*str*) – The path to sphinx's docs folder, where the tests will be run; by default `./docs/`
- **work_dir** (*str*) – The current Working Directory; by default
- **prechecks_done** (*bool*) – Whether the prechecks present in *~CanGraph.deploy.make_sphinx_prechecks* have already been made
- **custom_domain** (*str*) – A custom domain to deploy de docs to.

Returns Whether the prechecks have already been done; always True if the function is run

Return type bool

Note: For `custom_domain` to work, please configure your DNS records apparently

---

Note: `modules.rst` is not removed, but it is correctly ignored in conf.py

---

git_push(*path_to_repo*, *remote_name*, *commit_message*)

Pushes the current repo's state to a remote git repository

Parameters

- **path_to_repo** (*str*) – The path to the local `.git` folder
- **remote_name** (*str*) – The name of the remote to which we want to commit, which must be previously configured (see `CanGraph.setup.setup_git`)
- **commit_message** (*str*) – The Git Commit Message for the current repo's state

Note: `gitpython` is not good at managing complex commit messages (i.e. those with a Subject and a Body). If you want to add one of those, please, use `\n` as the separator; the function will take care of the rest

See also:

The approach taken hare was inspired by StackOverflow #41836988

`main`()

The function that executes the code

`make_sphinx_prechecks`(*docs_folder='./docs/'*, *work_dir='.'*)

Generates sphinx api-docs for automatic documentation and uses `make linkcheck`` to check for broken links

Parameters

- **docs_folder** (*str*) – The path to sphinx's docs folder, where the tests will be run; by default `./docs/`
- **work_dir** (*str*) – The current Working Directory; by default `.`

2.5 CanGraph.main module

A python module that leverages the functions present in the *miscelaneous* module and all other subpackages to annotate metabolites using a graph format and Neo4J, and then provides an GraphML export file.

For more details on how to run this script, please consult the package's README

`add_mesh_and_metanetx`()

`annotate_using_wikidata`()

`build_from_file`(*filepath*)

TODO EXPLICAR PQ HAGO LO DE RELPATH

`find_reasons_to_import`(*text*, *filepath*, *chebi_ids*, *names*, *hmdb_ids*, *inchis*)

`link_to_original_data`(*row*, *import_based_on*)

`main`()

The function that executes the code

Todo: CAMBIAR NOMBRE A LOS MESH PARA INDICAR EL TIPO. AÑADIR NAME A LOS WIKIDATA

Todo: FIX THE REPEAT TRANSACTION FUNCTION

Todo: Match partial InChIs based on DICE-MACCS

Todo: QUE FUNCIONE -> ACTUALMENTE ESTA SECCION RALENTIZA MAZO

Todo: CHECK APOC IS INSTALLED

Todo: FIX MAIN

Todo: MERGE BY INCHI, METANETX ID

Todo: Fix find_protein_interactions_in_metanetx

Todo: Mover esa funcion de setup a misc

Todo: EDIT conf.py

Todo: Detail Schema Changes

Todo: Document the following Schema Changes: For Subject, we have a composite PK: Exposome_Explorer_ID, Age, Gender e Information Now, more diseases will have a WikiData_ID and a related MeSH. This will help with networking. And, this diseases dont even need to be a part of a cancer! The Gene nodes no longer exist in the full db? -> They do

`scan_folder()`

2.6 CanGraph.miscelaneous module

A python module that provides a collection of functions to be used across the different scripts present in the CanGraph package, with various, useful functionalities

`call_db_schema_visualization(tx)`

Shows the DB Schema. This function is intended to be run only in Neo4J's console, since it produces no output when called from the driver.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Todo: Make it download the image

`clean_database()`

A CYPHER query that gets all the nodes in a Neo4J database and removes them, in transactions of 1000 rows to alleviate memory load

Returns A text chain that represents the CYPHER query with the desired output. This can be run using: `neo4j.Session.run`

Return type str

Note: This is an **autocommit transaction**. This means that, in order to not keep data in memory (and make running it with a huge amount of data) more efficient, you will need to add `:auto ` when calling it from the Neo4J browser, or call it as using `neo4j.Session.run` from the driver.

`create_n10s_graphconfig`(*tx*)

A CYPHER query that creates a *neosemantics* (n10s) constraint to hold all the RDF we will import.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

See also:

More information on this approach can be found in Neosemantics' 101 Guide and in Neo4J's guide on how to import data from Wikidata , where this approach was taken from

Deprecated since version 0.9: Since we are importing based on apoc.load.jsonParams, this is not needed anymore

`download`(*url*, *folder*)

Downloads a file from the internet into a given folder

Parameters

- **url** (*str*) – The Uniform Resource Locator for the Zipfile to be downloaded and unzipped
- **folder** (*str*) – The folder under which the file will be stored.

Returns This function downloads and unzips the file in the desired folder, but does not produce any particular return.

`download_and_unzip`(*url*, *folder*)

Downloads and unzips a given Zipfile from the internet; useful for databases which provide zip access.

Parameters

- **url** (*str*) – The Uniform Resource Locator for the Zipfile to be downloaded and unzipped
- **folder** (*str*) – The folder under which the file will be stored.

Returns This function downloads and unzips the file in the desired folder, but does not produce any particular return.

See also:

Code snippets for this function were taken from Shyamal Vaderia's Github and from StackOverflow #32123394

`export_graphml`(*tx*, *exportname*)

Exports a Neo4J graph to GraphML format. The graph will be exported to Neo4JImportPath

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **exportname** (*str*) – The name for the exported file, which will be saved under ./Neo4JImportPath/

Returns

A Neo4J connexion to the database that exports the file, using batch optimizations and smaller batch sizes to try to keep the impact on memory use low

Return type neo4j.work.result.Result

Note: for this to work, you HAVE TO have APOC availaible on your Neo4J installation

---

find_synonyms_in_metanetx(*driver*, *query_type*, *query*)

A function that automates finding synonyms in MetaNetX for a given metabolite, using *CanGraph.MeSHandMetaNetX.build_database.read_synonyms_in_metanetx*.

Parameters

- **driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use
- **query_type** (*str*) – one of ["HMDB_ID","ChEBI_ID","InChI","Name"]; the type of query we will be searching synonyms in
- **query** (*str*) – The value of an identifier based on which we want to find synonyms for a metabolite

Returns a list of the synonyms that MetaNetx can find for `query`, including the query itself

Return type list

get_import_path(*current_driver*)

A function that runs *neo4j_import_path_query* to get Neo4J's Import Path

---

Note: By doing the Neo4JImportPath search this way (in two functions), we are able to run the query as a :obj: read_transaction, which, unlike autocommit transactions, allows the query to be better controlled, and repeated in case it fails.

Parameters **current_driver** (*neo4j.Driver*) – Neo4J's Bolt Driver currently in use

Returns Neo4J's Import Path, i.e., where Neo4J will pick up files to be imported using the `file:///` schema

Return type str

import_graphml(*tx*, *importname*)

Imports a GraphML file into a Neo4J graph. The file has to be located in Neo4JImportPath

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **importname** (*str*) – The name for the file to be imported, which must be under ./Neo4JImportPath/

Returns

A Neo4J connexion to the database that imports the file, using batch optimizations and smaller batch sizes to try to keep the impact on memory use low

Return type neo4j.work.result.Result

Note: for this to work, you HAVE TO have APOC availaible on your Neo4J installation

---

neo4j_import_path_query(*tx*)

A CYPHER query able to find Neo4J's Import Path. When used in *get_import_path*, it returns a one-item list with the import path itself, but it **needs** to be executed in *get_import_path* for it to do so.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A one-item list containing the Neo4J Import Path

Return type str

purge_database(*driver*)

A series of commands that purge a database, removing unnecessary, duplicated or empty nodes and merging those without necessary properties This has been converted into a common function to standarize the ways the nodes are merged.

Args: driver (neo4j.Driver): Neo4J's Bolt Driver currently in use

Returns This function modifies the Neo4J Database as desired, but does not produce any particular return.

Warning: When modifying, take good care on how the keys names are written: with *remove_duplicate_nodes*, sometimes, if a key is not present, all nodes will be merged!

remove_ExternalEquivalent(*tx*)

Removes all nodes of type: ExternalEquivalent from he DataBase; since this do not add new info, one might consider them not useful.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

remove_duplicate_nodes(*tx*, *node_type*, *condition*, *optional_where=''*)

Removes any two nodes of any given `node_type` with the same `condition`.

Parameters

- **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running
- **node_type** (*str*) – The label of the nodes that will be selected for merging. If more than one, leave null and set optional_where = `WHERE (n:Label1 OR n:Label2) AND ...`
- **condition** (*str*) – The node properties used for collecting, if not using all properties. This can be expressed as: `n.{property_1} as a, n.{property_n} as b`
- **optional_where** (*str*) – An optional `WHERE` Neo4J Statement to constrain the nodes that are picked up; default is "" (no statement)

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Warning: When using, take good care on how the keys names are written: sometimes, if a key is not present, all nodes will be merged!

remove_duplicate_relationships(*tx*)

Removes duplicated relationships between ANY existing pair of nodes.

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Note: Only deletes DIRECTED relationships between THE SAME nodes, combining their properties

See also:

This way of working has been taken from StackOverflow #18724939

remove_n10s_graphconfig(*tx*)

Removes the "_GraphConfig" node, which is necessary for querying SPARQL endpoints but not at all useful in our final export

Parameters **tx** (*neo4j.work.simple.Session*) – The session under which the driver is running

Returns A Neo4J connexion to the database that modifies it according to the CYPHER statement contained in the function.

Return type neo4j.work.result.Result

Deprecated since version 0.9: Since we are importing based on apoc.load.jsonParams, this is not needed anymore

repeat_transaction(*tx*, *num_retries*, *session*, *bar*, *number=None*, *query='WikiData'*)

A function that repeats transactions whenever an error is found. This may make an incorrect script unnecessarily repeat; however, since the error is printed, one can discriminate those out, and the function remains helpful to prevent SPARQL Read Time-Outs.

split_csv(*filename*, *folder*, *sep=','*, *sep_out=','*, *startFrom=0*, *withStepsOf=1*)

Splits a given .csv/tsv file in n smaller csv files, one for each row on the original file, so that it does not crash when processing it. It also allows to start reading from `startFrom` lines

Parameters

- **filepath** (*str*) – The path to the file that needs to be xplitted
- **splittag** (*str*) – The tag based on which the file will be split
- **bigtag** (*str*) – The main tag of the file, which needs to be re-added.

Returns The number of files that have been produced from the original

Return type int

Warning: Original file will be removed

split_xml(*filepath*, *splittag*, *bigtag*)

Splits a given .xml file in n smaller XML files, one for each `splittag` section that is pressent in the original file, which should be of type `bigtag`. For example, we might have an <hmdb> file which we want to slit based on the <metabolite> items therein contained. Ths is so that Neo4J does not crash when processing it.

Parameters

- **filepath** (*str*) – The path to the file that needs to be xplitted

- **splittag** (*str*) – The tag based on which the file will be split
- **bigtag** (*str*) – The main tag of the file, which needs to be re-added.

Returns The number of files that have been produced from the original

Return type int

2.7 CanGraph.setup module

A python module that prepares the local environment, to be able to run the *main* function. This can be either run in an interactive way, requiring user input; or in a automatic way, in order to pre-configure things, for example, if you are using the singularity package

For more details on how to run this script, please consult the package's README

check_file(*filepath*)

Checks for the presence of a file, and exits the program

Parameters **filepath** (*str*) – The path of the file its existence is being checked

Raises **ValueError** – If the file does not exist

Returns True if successful, False otherwise.

Return type bool

final_message(*interactive=False*)

Prompts the user with a final message.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

initial_message(*interactive=False*)

Prompts the user with an initial message if the session is set to be interactive.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

install_packages(*requirements_file=None*, *package_name=None*)

Automates installing packages using PIP

Parameters

- **requirements_file** (*str*) – The path to a "requirements.txt" file, containing one requirement per line
- **package_name** (*str*) – A package to be installed

Raises **ValueError** – If neither a `requirements_file` nor a `package_name` is provided

main()

The function that executes the code

setup_drugbank(*interactive=False*)

Sets up the files relative to the SMPDB database in the "Databases" folder, splitting them for easier processing later on.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

---

Todo: What if `interactive=False`?

---

setup_exposome(*interactive=False*)

Sets up the files relative to the Exposome Explorer database in the "Databases" folder, splitting them for easier processing later on.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

---

Todo: What if `interactive=False`?

---

setup_folders(*interactive=False*)

Creates the "Databases" folder if it does not exist. If it does, it exits so as not to overwrite.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

Raises **ValueError** – If the Databases folder already exists (so as not to overwrite)

Returns True if successful, False otherwise.

Return type bool

setup_hmdb(*interactive=False*)

Sets up the files relative to the HMDB database in the "Databases" folder, splitting them for easier processing later on.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

setup_smpdb(*interactive=False*)

Sets up the files relative to the SMPDB database in the "Databases" folder, splitting them for easier processing later on.

Parameters **interactive** (*bool*) – Whether the session is set to be interactive or not

A utility to study and analyse cancer-associated metabolites using knowledge graphs

Get started

CanGraph is a python program that allows you to extract information about a newly discovered or existing metabolite from the following databases:

- Human Metabolome DB
- DrugBank
- Exposome Explorer
- WikiData
- SMPDB

- MesH and MetaNetX

For this purpose, CanGraph accepts any of the following inputs:

- **InChI:** The International Chemical Identifier for the metabolite, which can be calculated using Open Source Software and identifies a metabolite with 99.99% accuracy based on its structure
- **InChIKey:** The Hashed, shortened version of the InChI, sometimes used for efficiency
- **Name:** A commonly accepted name for the metabolite; preferably, IUPAC's standarized name
- **HMDB_ID:** The Metabolite's Identifier in the Human Metabolome Database
- **ChEBI_ID:** The Metabolite's Identifier in the ChEBI Database

which must be provided as explained in *CanGraph's README*

CHAPTER

THREE

ACKNOWLEDGEMENTS

CanGraph has been possible thanks to the International Agency for Research on Cancer's OncoMetaBolomics Team, which provided funding for the project

PYTHON MODULE INDEX

C

INDEX

A

B

C

D

E

F

G

I

L

M

N

P

R

S

W