Documentation for building the database

repository: https://github.com/BackofenLab/protein-graph-database/tree/workTillman

1. Folders

On the repository in the folder build_db there are several folders.

1.1 Original data

The methods, which are used to create the database, expect the original unedited data to be here.

1.1.1 exp_data

This is the in-house data. You can download it from here:

...

The methods expect the .csv files from the link here, so make sure to put them in this folder.

1.1.2 string_data

This is the data from string. I wrote a python script (get_string_data) to download most of it.

The only thing which does not get downloaded by the script is the file:
string_SYMBOL_ENSEMBL.tsv

This file is stored in the repository in the folder data as a gz file. You will have to unzip it and put it there.

1.1.3 functional_terms_data

This is partly data from string and partly inhouse data, which got generated by a bachelor student. The data from string gets downloaded by the pythonscript get_string_data.

The rest of the data is not yet provided online. Ask your supervisor where to get it. The data u need is the file OVERLAP.csv created by victor.

1.2 Edited data

This is the folder to which the methods write the data, after they filtered it.

1.3 Final data

If u create a toy example with the corresponding script the edited data will end up in this folder.

1.4 Python files

This the folder which contains the queries and python files to generate the database.

1.4 Documentation

This folder contains the documentation and a folder, which contains empty csv files with only the header. This is for reference on how to csv files have to look.

2. Python files

This part will only contain a short description what these files do. For further explanation look into the files.

2.1 get_string_files.py

This downloads data from the string website. Make sure u provide the correct links in the main function of the file. The website it downloads from is:

https://string-db.org/cgi/download?sessionId=blNmxwnnPdkJ&species_text=Mus+musculus

2.2 filter.py

This filters the data from the folder original data, in such a way that the edit files can use it properly.

2.3 edit_exp_data.py

This edits the exp data further.

2.4 edit_string_files.py

This edits the string data further.

2.5 run_files.py

This calls the files, which edit the data, so that u do not have to call them individually

2.6 create_toy_example.py

This creates a toy example from the edited data, so that when testing one does not work on the whole data.

2.7 upload_data.py

This uploads the data to the neo4j database. The neo4j database has to be running so that it works. It gives the user console feedback. You have to provide the port, the name and the password of the neo4j database in the main function of the file.

This writes and deletes files in the neo4j import folder. So make sure that this folder has corresponding rights.

3 Queries

These are the queries which get used by the upload_data.py file. They have to a certain structure which will be now described.

Some times a line is not needed of the query blueprint is not needed in that case there will be a place holder. The place holder looks like this:

"MATCH (n) RETURN n LIMIT 0"

All the lines of the queries have to be written with “ at the start and the end!

The queries use apoc. Look in the repository for the file Database_Doc_Beta.pdf and follow the instructions to install apoc.

Explain how to use apoc here.

3.1 queries_correlation.csv

Example:

"# Upload the OR and Protein correlation"

"create (s:Association_peak_tar{Study: 1, Cell: 'Microglia'});"

"Load csv With HEADERS from 'file:///temp.csv' as map Create (t:Association_temp) set t = map;"

"Match (a:Association_temp), (p:Protein),(s:Association_peak_tar) Where (toLower(p.SYMBOL) = toLower(a.SYMBOL)) Create (s)-[h:Protein_OR_correlation]->(p) set h=a;"

"Match (a:Association_temp), (t:OpenRegion{nearest_index: a.nearest_index}), (s:Association_peak_tar) Create (s)-[h:OR_Protein_correlation]->(t) set h=a;"

"Match (n:Association_peak_tar),(s:Study{Nr:n.Study,Cell:n.Cell}), (c:Cell{Study:n.Study,Cell:n.Cell}) Create (s)-[j:study_is_about]->(n)<-[a:cell_is_about]-(c);"

"match(a:Association_temp) delete a;"

The 1 line is the comment which will give the user feedback when he calls the python file.

The 2 line creates the source/correlation node, which connects the entities who are correlated. Here the study and cell type is set.

The 3 line loads the csv with the corresponding relations and their attributes. Then it creates a temp node for every relationship. It is done in this way because it is way faster to create nodes and then match them, to create a relationship, instead of directly uploading the relationships.

The 4 line creates the first relationship by matching the temp nodes and the real nodes and then creating the corresponding relationship. The where part defines on what attributes of the correlations u are matching. And the to lower makes it case insensitive.

The 5 line creates the second relationship by matching the temp nodes and the real nodes and then creating the corresponding relationship.

The 6 line creates the relations between the study and cell and the correlation node.

The 7 line deletes the temporary nodes again

3.2 queries_biological_entities.csv

"# Upload the Transcriptionfactors"

"CALL apoc.load.csv('temp.csv') YIELD map Create (t:TranscriptionFactor) Set t = map;"

The 1 line is the comment which will give the user feedback when he calls the file.

The 2 line loads the csv with the corresponding entity nodes and their attributes and then uploads them into the database.

3.3 query_study_cell.csv

Creates Study and Cell node

4 Structure of the original data

The original data needs to be in a certain format. Look in the folder data_structure to find the blue prints of all the original files.