Final assignment: scoring network alignments

GOal

In this assignment you will need to combine all that you have learned over the past couple of weeks. So if you need to, it’s a good idea to take a look at the previous assignments. In this assignment you are given three alignments of proteins of mus musculus ("mmu"), rattus norvegicus ("rno") and homo sapiens ("hsa"), which have been obtained through alignment of the corresponding protein-protein interaction networks. Since you have no clue about the performance of the used network alignment algorithm, you want to validate these alignments. One way of doing this is by computing a score based on shared GO terms: if the alignment is good then aligned protein pairs should have a lot of common GO terms.

Assignment

Start by downloading and unzipping the file “Final assignment.zip” from Canvas. When unzipped you should see three directories: "alignments", "GO" and "mapping". The zip file also contains the skeleton script you should use. Let’s take a look at the first directory. Open it and you should see three files. Open one with your favorite text editor. For instance, these are the first lines of "rno-hsa.sif":

```
ENSRNOP00000016077 ENSP00000349465
ENSRNOP00000000009 ENSP00000304169
ENSRNOP00000013694 ENSP00000265165
ENSRNOP00000000025 ENSP00000014930
ENSRNOP00000044240 ENSP00000350766
```

So every line consists of a pair of aligned proteins. The first line of the file rno-hsa.sif reproduced above tells us that the protein "ENSRNOP00000016077" in rats is aligned with "ENSP00000349465" in humans. From the filename we know to which two species the columns correspond. In this particular case the first column corresponds to "rno" (rats) and the second to "hsa" (humans). Some of you might also recognize that Ensembl protein IDs are used for both species' proteins.

Now let’s take a look at one of the files in the directory "GO". The files there are rather big (30-50 MB). That is because these files correspond to database dumps of Gene Ontology annotations. Let’s take a closer look at "rno.go". You can see that the file starts with a descriptive header – every line of this header starts with "!". The actual data entries look like this:

RGD 1302933 Mcpt1l3 GO:0003824 RGD:1600115 IEA InterPro:IPR009003...
RGD 1302933 Mcpt1l3 GO:0004252 RGD:1600115 IEA InterPro:IPR001254...

What is important to us in this assignment are columns 2 and 5 (column 4 is empty), which correspond to the protein ID and GO-term ID, respectively. So the first line above tells us that the protein with ID "1302933" is annotated with the GO term "GO:0003824". In the second line we can also see that the same protein also has GO term "GO:0004252".

Unfortunately, this protein ID is of a totally different format than the Ensembl protein IDs we use in the alignment files. To make matters worse, if you take a look at the last lines of the same file then you can see yet another ID format:

UniProtKB Q9Z340 Pard3 GO:0019903 PMID:18550519 IPI UniProtKB:O54857...

A warm welcome to identifier hell! This is something that you will encounter a lot – every time a group develops something new they just love to introduce a new set of identifiers. In order to deal with this mess, we need to map all these IDs to their Ensembl counterparts. That is why we have three mapping files in the directory mapping. Let's open "rno.map":

Ensembl_Protein_ID UniProt/SwissProt_Accession UniProt/TrEMBL_Accession RGD_ID
ENSRNOP00000000008 P18088 C9E895 2652 ENSRNOP00000000008 P18088
B3VQJ0 2652

Let's explain what the lines above mean: IDs "2652", "C9E895", "B3VQJ0" all map to Ensembl ID "ENSRNOP00000000008".

The goal of the exercise is to score an alignment based on shared GO terms. More precisely, we do this using the following calculation:

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

This is also known as the *Jaccard index*, hence the use of the letter J. In this particular case A and B are sets of GO terms belonging to two aligned proteins.

Time for an example. Assume you have the following (fictional) alignment file:

```
a x
b y
```

So protein "a" is aligned with "x" and protein "b" is aligned with "y". Now let's just say that after dealing with the ID mess, we end up with the following list of GO terms per protein:

a GO:1 GO:2
b GO:2 GO:3 GO:5
x GO:1 GO:5
y GO:2 GO:3 GO:9 GO:10

The score of this alignment would then simply be the sum of the scores of (a, x) and (b, y). Let’s start with the Jaccard index of "a" and "x". First we need to count how many terms "a" and "x" have in common, which is 1 (only "GO:1"). Then we need to count many different terms they have in total: 3 ("GO:1" is only counted once!). So the contribution of "a" versus "x" to the alignment score is 1/3 = 0.33. Similarly, you can compute the contribution of "b" and "y" to be 2/5 = 0.4. So the total alignment score is 0.4 + 0.334 = 0.734.

The assignment is to write a script "score.py" which takes five command line arguments as input:

```
./go_score.py <SIF> <GO1> <GO2> <MAP1> <MAP2>
```

Here *<SIF>* is an alignment file, *<GO1>* and *<GO2>* are GO annotation files, *<MAP1>* and *<MAP2>* are mapping files. The output produced by your script should contain the alignment score. You need to hand in the assignment on CodeGrade.

```
./go_score.py alignments/rno-mmu.sif GO/rno.go
GO/mmu.go mapping/rno.map mapping/mmu.map
./go_score.py alignments/rno-hsa.sif GO/rno.go
GO/hsa.go mapping/rno.map mapping/hsa.map
./go_score.py alignments/mmu-hsa.sif GO/mmu.go
GO/hsa.go mapping/mmu.map mapping/hsa.map
```

Don’t forget to do some error handling in your script! At the very least, you should check whether the number of arguments you get via the command line is correct.

To help you some more with the assignment, in the following subsections the steps that you need to take are described in more detail.

Making the mapping dictionaries

The goal of this subsection is to write a function `get_mapping(map_file)` whose only argument `map_file` is the name of the file containing the IDs. So in our case `map_file` will be one of "mapping/rno.map", "mapping/mmu.map" and "mapping/hsa.map". These files have respectively **four, two and three columns**. The first column always corresponds to the Ensembl IDs we use. The `get_mapping` function should return a list of dictionaries. Every dictionary should have as a key a non-Ensembl ID and as value the corresponding Ensembl ID. The number of dictionaries in the list should be equal to the number of columns minus one.

The skeleton script contains a start to the function `get_mapping(map_file)`. Finish the script by filling in the for loop. Let's look at an example. Suppose `map_file` contains the following:

Ensembl_Protein_ID UniProt/SwissProt_Accession UniProt/TrEMBL_Accession RGD_ID ENSRNOP00000000008 P18088 C9E895 2652 ENSRNOP00000000008 P18088 B3VQJ0 2652

Then `get_mapping(map_file)` should return a list containing three dictionaries. The first dictionary should contain only one entry with key "P18088" and value "ENSRNOP00000000008". The second dictionary should contain two entries with keys "C9E895" and "B3VQJ0" mapping again to "ENSRNOP00000000008". The last dictionary, which is the third element of the list, should contain just one entry with key "2652" and again the same value.

Hint: This how you append a dictionary `d` (or any other type of element) to list `l`:

```
>>> l = [1, 2, 3]
>>> d = {}
>>> l.append(d)
>>> l
 [1, 2, 3, {}]
```

This function should return a list containing dictionaries for each different database.

Parsing the GO file and getting the terms

The next step is to get the GO-terms out of the GO file. The function `get_go_terms(mapping_list, go_file)` whose first argument corresponds to the list of dictionaries computed by `get_mapping` and second argument corresponds to the name of a GO file. The result of this function should be a dictionary indexed by Ensembl protein IDs and having as values sets of GO terms.

So you are probably thinking: why sets, why not just lists? This will become apparent in the next subsection. A set is kind of similar to a list, the biggest difference is that does not allow for duplicates. Try doing the following in the Python shell.

```
>>> s = set()
>>> type(s)
>>> s.add("appel")
>>> s.add("banaan")
>>> s
>>> s.add("appel")
>>> s
>>> s.add("peer")
>>> s
```

More information: sets

So when you try to add something to a set that is already in that set, nothing will happen and the set will remain unaltered. This is exactly what we want, we are already in a messy business with all those ID mappings, and because of that we might end up with duplicate GO terms for a particular Ensembl protein ID. With sets we prevent this.

Note: Another difference between `set()` and `list()` is that the ordering of the elements in a set is *not* explicitly specified. If you use a `for` loop you may loop through elements in a different order than you put them in, for example.

Let's look at an example. Suppose go_file contains the following.

UniProtKB P10070 GLI2 GO:0045944 PMID:16553965 UniProtKB P10070 GLI2 GO:0045944 PMID:12165851 UniProtKB P10070 GLI2 GO:0009913 PMID:12165851

Column 2 contains the protein ID, in this case UniProt ID, and column 5 the GO term. In this example UniProt protein ID "P10070" corresponds with Ensembl ID "ENSP00000354586" in `mapping_list`. So the returning dictionary should contain the key "ENSP00000354586" with GO terms "0045944" and "0009913" in a set as the value.

The skeleton script contains a start to this function. Finish this script so that the result is a dictionary with for each Ensembl ID the corresponding GO terms as a set.

Hints:

- Don't forget about split(), entries are separated by a tab-character.
- Skip lines that start with "!" as they are just comments.
- Protein IDs are in column 2 and GO terms in column 5. Remember the way Python list numbering works!
- Given a protein ID (from the GO file), you can look up the corresponding Ensembl ID by checking all the dictionaries in mapping_list with a for loop.

Computing the score

In this section we'll implement the function `compute_score(alignment_file, go_one_dict, go_two_dict)`. The first argument is the name of the alignment file. The second and third arguments correspond to two dictionaries that have been generated by the function `get_go_terms`.

Here the fact that we generated sets in the previous function really pays off! Remember that the determination of the Jaccard index is done in the following fashion:

$$J(A,\ B)\ = \frac{A \cap B}{A \cup B}$$

Here, as you know A and B are sets, and we are supposed to divide the *set intersection* by the *set union*. Lucky for us, Python, being a convenient language, implements both these operators:

```
>>> farm_animals = {"dog", "cow", "sheep", "horse"}
>>> house_animals = {"dog", "cat", "gerbil",
"goldfish"} >>> farm_animals & house_animals
set(['dog'])
>>> farm_animals | house_animals
set(['sheep', 'horse', 'gerbil', 'cow', 'dog', 'cat', 'goldfish'])
```

Note: Just like you can assign a three-item list to a variable by the *list literal* `[1, 2, 3]` you can also assign a set instead by using curly brackets `{}` instead of square brackets `[]`. The only time this won't work is for the empty set `set([])`, since dictionaries are also specified by curly brackets and Python won't know whether you want an empty set or an empty dictionary in such a case.

Finish the skeleton script by computing the Jaccard score for the alignment. Also keep track of the number of unmapped proteins. Return those scores to the `main()` function and print the scores.

Combining everything together!

Now only the easy part remains. You just need to call the correct functions in the proper order. So first we construct the two mapping lists (each containing several dictionaries) by calling the function `get_mapping` on the two specified map files. Then we use these mapping lists as arguments to the function `get_go_terms`. The result of that function is the input of the `compute_score` function.