**GenBank file parser and database connection**

Documentation for files parser.py and split\_file.py.

**Description of project**

This tier of the project aims to parse a GenBank file and store some relevant information in an SQL database.

**Setup: split\_file.py**

Some steps must be taken prior to use of the parser. Please see the 'Dependencies' section below for required imports. As well as these, it is important to note that the split\_file.py script must be run before the parser is used. The file that the user passes into it must be a GenBank format file. The split\_file.py script takes the raw GenBank file and splits it at the '//' entry separator, thus, a separate file for each gene locus is generated. Ensure that these files are stored in a directory with nothing else in it. Once this step has been completed, set the 'indir' variable in the parser.py script to the directory containing your split files.

**Dependencies: parser.py, split\_file.py**

The following imports must be performed in order to run the code. Code to import them is included in both parser.py and split\_file.py

split\_file.py:

import re

parser.py:

import re

import os

import pandas as pd

import mysql.connector

from sqlalchemy import create\_engine

**Data extraction tier: parser.py**

**Functions**

*First core function: numerical\_split*

Allows string to digit conversion of the filenames derived from the split GenBank file.

Parameters:

1. filename

*Second core function: match\_finder*

Finds one match to the supplied regex in each separate file derived from the split GenBank file.

Parameters:

1. list - this is an empty list in which you want to store the

function output

1. compiler - this is the regex compiler in the format:

compiler = re.compile(r" regex here ", re.MULTILINE|re.DOTALL)

1. else\_statement - this is the value that will be appended to the

list specified in parameter 1. if no value can be found by the regex in the GenBank file. DEFAULT = 'None'

*Third core function: findall\_matcher*

Finds all matches to the supplied regex in each separate file derived from the split GenBank file.

Parameters:

1. list\_ - this is an empty list in which you want to store the

function output

1. pattern - this is the findall regex pattern in the form r"pattern"

**Implementation: parser.py**

Here, the functions are used to extract useful parts of the GenBank file.

Regular expressions have been designed to extract key data the match\_finder and findall\_matcher functions store their captured data in the following lists:

|  |  |
| --- | --- |
| List name | Description |
| gene\_ids | Unique locus identifier |
| genbank\_accessions | Unique genbank identifier |
| gene\_name | Name of the gene |
| clean\_dna\_seq | Nucleotide sequence of the gene |
| gene\_products | This list contains the 1st protein product of the gene |
| chr\_loc | Chromosomal location of the locus |
| clean\_protein\_seq | Amino acid sequence encoded by the coding sequence of the gene |
| exon\_start | Contains sub-lists of exon start positions for each locus |
| exon\_end | Contains sub-lists of exon end positions for each locus |

**Data processing: parser.py**

Some of the data requires additional post-capture processing

DNA sequence:

cleaning of whitespaces and digits present in the GenBank file – begins with list dna\_seq and the database-friendly data is stored in the list clean\_dna\_seq.

Protein sequence:

Cleaning of whitespaces

Coding sequence:

cds\_grab is the initial list containing the raw strings

It then undergoes whitespace stripping, and the data is contained in cds\_ws\_strip.

The list then becomes the new list: cds\_ws\_strip. Extra characters are then stripped, resulting in the list clean\_boundaries.

Then the items (now in the form '123..456') are split at the ‘..’.

After this, exons that span multiple genes are removed.

The list is now called remove\_spans. The start and end positions of the coding sequence are then stored in the lists exon\_start and exon\_end.

**Preparing the data for database import: parser.py**

The tables of the database are fed the following data:-

Table 1

Coding\_region:

- gene\_ids as 'Gene\_ID'

- exon start as 'Start\_location'

- exon end as 'End\_location'

Table 2

Gene\_info

- gene\_ids as 'Gene\_ID'

- chr\_loc as 'Chromosome\_location'

- clean\_dna\_seq as DNA\_sequence

- clean\_protein\_seq as Protein\_sequence

- gene\_products as Protein\_product

Preparing exon\_start and exon\_end:

A zip object called zipped\_id\_start\_end is created - it is a list of tuples. A list comprehension is utilised to go from lists of lists of the form exon\_start, exon\_end to a list of tuples with repeating gene\_ids i.e. any one gene\_ids entry will be contained in n rows where n is the number of start positions and corresponding end positions.

Removal of splice variants:

Splice variants are then identified and deleted. An enumerating for loop identifies the gene\_ids associated with these, and removes any splice variant tuples from the zipped\_id\_start\_end list and any rows containing splice variants from the Gene\_info table.

Preparing the data for table 1:

Using pandas (imported as pd), the zip object is converted to a Pandas DataFrame.

Preparing the data for table 2:

Using pandas (imported as pd), the five lists named under 'Table 2' above are

converted to a Pandas DataFrame object.

A connection to the database is generated using the sqlAlchemy function

create\_engine.

mysql.connector is then used to login into the database and port the dataframes in.

**The database**

**Physical schema**

Here, the relationship between the tables is explained. The Coding\_region table uses the Gene\_ID primary key from the Gene\_info table as its foreign key. The Coding\_region table contains duplicate keys. Thus, for each key in the Gene\_info table, there could be zero or more entries in the Coding\_region table. The field sizes are shown in brackets.

../../PycharmProjects/BiocomputingII/databaselayer/db_schema_30:04.pdf

**Table definitions**

