GEO Dataset Tools Documentation

Michael A. Gohde

October 24, 2017

1 Introduction

The GEO Dataset Tools provide a set of utilities and functions that should make it relatively straightforward to query the NIH's GEO database. Ultimately, the information gained from these tools should make it easier to download datasets and to collect information about how they were created.

Currently, the tools present here are in a state of fairly heavy development, so any discrepancies between that which is documented here and the actual operation of the tools is likely due to recent changes to how they were designed.

2 Querying the GEO Database for Entries

The first step in the process of finding specific datasets to work with is to actually determine which datasets exist. In the project directory, there exists a script named, 'query_geo.py'. Its purpose is to search the GEO dataset for various terms related to the protocols used for data collection and analysis. For example, it can be used to determine which entries in the GEO database were generated using GRO-Seq.

When the script is invoked, it queries the GEO database with the specified search term or, "gro-seq" if none is specified. If the query is successful, it will proceed to either print or write a file containing a set of database element IDs corresponding to elements matching the query. With this set of IDs, it will be possible to invoke another script to fetch all of the found elements from GEO. This process will be detailed in the next section.

2.1 Invoking query_geo.py

At the time of this writing, query_geo can be invoked in the following ways:

```
Usage: ./query_geo.py <args> query
Query the GEO database for numbers usable by fetch_groseq.py
If the -o switch is not specified, it is recommended that this program's output
be piped into either another command or a file.

<args> may be one of the following:
```

```
-h, --help      Prints this message.
-o=<filename>   Writes to an output file instead of stdout.
-v      Verbose output. All logging messages are written to stderr.

Examples:
./query_geo.py gro-seq >out.txt
./query_geo.py -o=out.txt pro-seq
```

Generally, the query parameter will correspond to a specific protocol, such as pro-seq, gro-seq, gro-cap, and 5'gro. In fact, the database fetching software detailed later in this document expects that this is the case. However, it should be possible to query the database for other features of interest, such as specific author names, species, etc.

It should also be noted that all this script works best when its output is redirected to a separate file. This makes it easier to view debugging information when the '-v' parameter is specified. Also, the other scripts in this project expect to be able to read a file instead of receiving GEO database IDs from the command line.

Below is an example session used to demonstrate a series of queries made to the GEO database:

```
user@computer ~/geo_dataset_tools $ ./query_geo.py -o=groseq.txt -v gro-seq
[Message] About to make query...
[Message] Query successful!
[Message] Found 1373 elements.
[Message] Done.

user@computer ~/geo_dataset_tools $ ./query_geo.py -o=proseq.txt -v pro-seq
[Message] About to make query...
[Message] Query successful!
[Message] Found 106 elements.
[Message] Done.

user@computer ~/geo_dataset_tools $ ./query_geo.py -o=5gro.txt "5'gro"
```

Note that the last command didn't print anything to the console. This is so because the '-v' parameter was not specified.

Contents of an the '5gro.txt' file generated above:

```
QUERY 5'gro
200068677
200090035
200083108
200045914
301678910
301678908
302396016
302396015
```

```
302193123
301119600
301119599
```

3 Generating a local metadata database

Once one or more sets of IDs has been obtained following the procedure outlined in the previous section, it's necessary to generate and maintain a local cache of information from the GEO database. This is done for a number of reasons:

1. It's significantly faster than repeatedly querying an online database.
2. Keeping a local cache reduces the load on the GEO database servers.
3. Having a local database following strict formatting rules will enable more applications to be developed in the future.
4. Additional data can be associated with each element, such as data series matrices, etc.
5. This enables consistency in implementation and usage.

In order to create this database, all that's necessary is to run the following command for all of the ID files fetched in the previous section:

```
user@computer ~/geo_dataset_tools $ ./make_groseq_database.py groseq.txt
    proseq.txt 5seq.txt grocap.txt db
```

In the above example, the local database creation tool will read through every ID defined in all of the files specified, then store appropriate metadata in the directory specified. It is possible to specify any number of input files (as long as there is at least one), and more queries can be integrated into the same database directory after it is populated for the first time. In fact, it should also be possible to merge the same query back into the database at a future date. Doing so will allow any changed elements to be added, while existing elements will be updated if changes have been made.

Full usage statement for the make_groseq_database script:

```
Usage: ./make_groseq_database.py idfile(s) dbdir
Fetches project summaries and adds appropriate metadata to a database of GEO
GRO-Seq data.
WARNING: This script may generate several thousand directories under dbdir.
```

4 Querying the local metadata database

Once enough metadata has been successfully fetched, it is possible to perform operations involving that metadata. In order to do so, it is necessary to invoke the 'query_groseq_database.py' script in the project directory.

The script's usage statement is listed below:

```
Usage: ./query_groseq_database.py [-s] dbdir command <args>
Query a GRO-Seq metadata database fetched with make_groseq_database.py
If -s is specified, then only series IDs will be reported on
If -pt=<comma separated list of protocols> is specified, then only IDs
     with a specific protocol will be reported on.

List of commands:
  listprotocols -- List all protocols in the current database.
  queryprotocol -- Print all elements matching a given protocol.
  protocoloverlap -- Print out the set of elements that overlap between
       different protocols.
  listspecies -- Print a listing of all species defined in the database.
  findspecies <species name in quotes> -- Find all projects that match a
       given species.
  getsummary <list of id numbers or paper names> -- Retrieves a summary for
       a given data element.
  fetchmatrices <id> -- Downloads data matrices necessary to fetch data for
       a set.
  fetchspmats <species name> -- Fetches all matrices for a given species.
  fetchallmatrices -- Fetch as many matrix files as possible. This will be
       slow!
  getsralist <id or paper name> -- Retrieves all SRAs for a given element
       given that matrices are present.
  getreadytosra -- Retrieves a list of all projects with fetched matrix
       files.
  getreadytodownload -- Retrieves a list of all projects that can be
       downloaded immediately.
  download <id or paper name> <outputdir> -- Downloads data into the
       specified directory
```

4.1 An explanation of the '-s' and '-pt' command line arguments

As documented, every command implemented in 'query_groseq_database' is designed to work with one of the following sets of elements:

1. Every element present in the database.
2. A specific set of elements explicitly listed by the user.
3. A specific element explicitly listed by the user.

The first case in particular is extremely limited in that it will often present quite a bit of unwanted data for every given query. For example, a query based on species name will include both collections of data and every entry for every data element in each collection.

The '-s' and '-pt' command line arguments reduce the size of the input set that query commands work on by allowing the user to specify specific additional attributes for what they would like to find. As such, the '-s' command limits all queries to just the set of defined collections, rather than the elements making them up. The '-pt' command similarly allows users to search for data collected under specific protocols.

Eventually, support for more flags will be implemented so that users can easily pass the results of one query into another.

4.2 listprotocols

This command lists the set of protocols defined in the database. This is actually the list of queries fed into the make_groseq_database script. At the time of this writing, this command effectively just lists the contents of the database directory specified.

4.3 queryprotocol

This command fetches a list of all elements matching a given protocol. This list includes a set of ID numbers and human-readable titles provided by the GEO database.

4.4 protocoloverlap

This command causes query_groseq_database to attempt to find which elements are the same across protocols. Certain data series defined by GEO may have multiple protocols associated with them. Running protocoloverlap should make it easier to find out which datasets have which protocols associated with them. In the future, more commands like this one will be implemented.

4.5 listspecies

This command does as specified. It searches for and returns a list of all species defined in the dataset sorted by the frequency in which they were mentioned. At the time of this writing, it is noted that certain data series may specify multiple species. This situation still must be rectified.

4.6 findspecies

This command searches for all IDs matching the specified species. At the time of this writing, multiple species may be defined per data element. This situation will be resolved or otherwise noted through program behavior in the future.

4.7 getsummary

This command prints a detailed summary for every ID or paper name specified. Individual IDs are to be separated by spaces. Eventually, wildcard and similar match support should be implemented.

4.8 fetchmatrices

This command fetches a set of data series matrices for the appropriate ID if they are defined in that ID's metadata. Please note that this command requires that series matrices be defined for the given data element. While almost every element has an associated matrix, you can check for whether one can be fetched by checking for the "Data matrix URL defined?" field on running the getsummary command. If "YES" is printed next to the field, then matrices can be fetched. Otherwise, no data matrices have been defined for the current element and thus cannot be fetched.

One side effect of having fetched series matrices for a given element is that every future listing for that element by other commands will contain a "paper name." This is a string consisting of the last name of the first contributor to the element's project along with the date that the element was posted to GEO. Please note that publication years for papers and the dates at which their associated data was submitted to GEO may differ.

4.9 fetchspmats

This command was implemented to make it easier to fetch all data matrices for a given species. It is equivalent to running the 'findspecies' command, then running 'fetchmatrices' for every result returned.

4.10 fetchallmatrices

This command attempts to fetch every series matrix file associated with every element in the database. Please note that due to the intensive nature of this command, it may take a very long time to complete and lacks much formal validation.

4.11 getsralist

If data series matrices have been successfully fetched for a given element, then this command looks up the SRA ID numbers for every data file associated with that element.

4.12 getreadytosra

This command is a convenient way of listing every project for which series matrices have been fetched.

4.13 getreadytodownload

This command lists all elements on which 'getsralist' has been run.

4.14 download

This command downloads the set of SRA files found by the 'getsralist' command into the directory specified on the command line.