Sequence similarity network and MetaSSN tutorial

John Chen, Jul 2020

Table of Contents

1. Overview 2

2. Background 2

3. Software requirements and set-up 3

4. Constructing an SSN 4

   4.1 Workflow 4

   4.2 Guide on SSN construction 5

      4.2.1 Data collection 5

      4.2.2 Data curation 7

      4.2.3 Merging data 8

      4.2.4 Data preprocessing 10

      4.2.5 All by all BLAST 12

5. Using MetaSSN to condense and annotate an SSN 14

   5.1 Workflow 14

   5.2 Collecting and curating annotation data 14

      5.2.1 Count of known sequences 15

      5.2.2 Label of known sequences 17

      5.2.3 Frequency of host taxonomies 18

      5.2.4 Distribution of sequence lengths 19

      5.2.5 Final overview 19

   5.3 Configuring MetaSSN 20

      5.3.1 Environment setup 20

      5.3.2 BLAST result input and MetaSSN output 21

      5.3.3 CD-hit cluster files 22

      5.3.4 Annotation files 23

   5.4 Running MetaSSN 25

      5.4.1 Run MetaSSN 25

      5.4.2 Examining the output 26

   5.5 Post-MetaSSN annotations 27

5.5.1 Updating old networks with new annotations 27
5.5.2 Mapping nodes in networks with higher node clustering cut-offs to networks with lower cut-offs 27
6. Network manipulation 28
6.1 Analysis using Cytoscape 28
6.1.1 Loading networks into Cytoscape 28
6.1.2 Selecting nodes 33
6.1.3 Adding annotations to selected nodes 39
6.1.4 Applying visualizations based on annotations 41
6.2 Network manipulation 44
6.2.1 Selecting subnetworks in CytoScape 45
6.2.2 Hiding and deleting nodes in CytoScape 46
6.2.3 Examining networks using node information 46
6.2.4 Extracting subnetworks at different cut-offs by node membership 48
6.3 Extraction of sequences from the network 49

1. Overview

This tutorial seeks to document the overall process of generating a sequence similarity network (SSN) and the processing of an SSN through the metaSSN python script. The tutorial assumes the user has conceptual understanding of what an SSN is and has a basic idea of what purpose the user wishes to use the SSN for. The focus of the tutorial is to address the technical requirements in constructing and annotating an SSN using an example data set.

The user will need to be at least somewhat familiar with running programs from the command line, as the tutorial almost exclusively uses command line based commands. The tutorial also makes use of Python scripts, also run from the command line. Since the tutorial focuses on the practical considerations of SSN generation rather than the exact details of each program being used, the user is encouraged to check the manual or help prompts of each software if they wish to learn more about the software specifically. All command line programs (BLAST, CD-hit and custom Python scripts) will have instructions and help prompts that are accessible with the '-h' argument for the command. The exception to this are instructions relating to the usage of metaSSN its self, which does not expose all arguments and instead relies on a configuration file (also see the metaSSN manual).

2. Background

The key advantage of the SSN is that it can display the interconnectivity of thousands of protein sequences or more, allowing for the viewing of a protein family's 'topology' in terms of how similar each member of the family is to one another. An SSN is a graph that shows the interconnections between a family of protein sequences. Nodes in the graph represent protein sequences and edges between nodes indicates that the connected nodes are related by homology (measured in BLAST bitscore) to a certain degree; the degree of connectivity can be adjusted by limiting the connections to only be present for higher bitscores. One limitation is the exponential nature of the data required to generate an SSN. To obtain all pairwise homology measures (to be used as edges), an all-by-all BLAST of all members in the protein family is conducted, making a maximum possible edge count of the number of sequences times the maximum BLAST output limit (default 500 hits per sequence). It is easy to get more edges than can be displayed in a network viewing application such as Cytoscape (above 500 thousand to 1 million edges), making such networks difficult to process for further analysis. The solution to overcome this limit is to cluster nodes connected by edges above a certain homology threshold into 'meta-nodes' (see the SSN Pipe software on GitHub: https://github.com/ahvdk/SSNpipe), drastically reducing the number of nodes and hence the number of edges between them. However, with such an approach, what use to be individual sequences in the SSN are now compiled into the same node. The annotation of nodes containing multiple sequences requires addition computational processes, which are what the metaSSN set of scripts seeks to provide.

3. Software requirements and set-up

The overall process of building an SSN requires two pieces of public software, which are BLAST and CD-hit. All by all BLAST requires a local installation of BLAST, while CD-hit can be performed on a web server or using a local copy (requires linux command line to install, available by default on linux/mac and can be downloaded in Windows as the 'ubuntu subsystem').

BLAST:

Info - https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download

Download - https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/

CD-hit:

Webserver - http://weizhong-lab.ucsd.edu/cdhit-web-server/cgi-bin/index.cgi

Download - https://github.com/weizhongli/cdhit

Additionally, I have provided a set of Python scripts that will simplify some of the steps. These can be substituted with the user's own scripts or processes if desired. However, if one wishes to follow the tutorial exactly, they will need a python 3.7 or higher installation to run the scripts. No additional Python dependencies are required.

The scripts need to be run from the command line in the following fashion:

```
python    name_of_script.py   input1    input2    etc.
```

Each script has it's own set of usage instructions, which can be accessed by typing the command without any input, or with the input '-h', as shown below:

```
python    name_of_script.py    -h
```

In the above examples it is assumed that the Python scripts are in the same folder as where the command is being given. To use the command in a different directory, copy the script of interest as well as the script 'helper.py' into the new folder.

For more advanced users, one can set up the files to be used from any where in the file system, and without the 'python' part of the command. For example:

```
name_of_script.py    input1    input2    etc.
```

To do this, add the folder containing the scripts to the PATH variable of your operating system.

Windows - https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/

Mac - https://hathaway.cc/2008/06/how-to-edit-your-path-environment-variables-on-mac/

Also, you will need to set the default application to open the '.py' file type as your python.exe. In Windows this can be achieved by right clicking on the file, choosing 'open with', then choosing 'Choose another app'. The equivalent method on mac is to right click, select 'Get Info' then 'Open With. This should open a file selection dialogue that asks what program you would like to open the file type with, use this to navigate to where your python is installed and select the 'python.exe'.

Some bash scripts will also be mentioned here and there, performing some convenience functions. These are not necessary to follow the tutorial. However, should one wish to use them, simply type the commands in the linux/mac/windows Ubuntu subsystem command line in the correct directory.

4. Constructing an SSN

This section seeks to detail the process of generating an SSN. The procedure encompasses all the requirements for a conventional SSN without any post-processing, but the tutorial does take special considerations to make the output convenient for the use of MetaSSN as a post-processing step.

4.1 Workflow

The overall process for SSN construction is as follows:

1. Data collection
2. Data curation
3. Merging data
4. Data preprocessing
5. All by all BLAST

The steps are described in more detail below.

1. Collection of data from a set of sources that comprises either a complete or representative set of the protein sequences in a family.

    a. Usually, most sequences do not have any reliable annotations (metadata) associated with them and sequences with known properties need to be specifically included in this, such as members with a confirmed structure or function.

2. For each individual data set, perform some curation for length and redundancy. This should be performed separately for each set of data, especially if they have different requirements—for example, due to different starting redundancy. The main goal is that only sequences that pass curation will be part of the SSN, and those that fail curation will be excluded.

    a. Sequences with extremely short and long lengths relative to the desired family should be excluded. Extremely short sequences often represent fragments of full sequences, while extremely long sequences may be multidomain proteins that are not of immediate interest when searching just for a single domain.

    b. Redundancy can be reduced by performing CD-hit at a relatively high cut-off of 95-100%. You may wish to skip the reduction of redundancy if you absolutely want every sequence to be included (which is only the case if you have prior knowledge that even a cd-hit at 100% will disrupt the process), but it may be better to reduce the redundancy and extract them from this clustering information afterwards.

3. All data to be considered for the SSN should be merged into the same Fasta file.

4. Before BLAST, the merged data set can be further processed to facilitate the downstream steps.

    a. You can convert the headers in the fasta file to numbers and keep the connection between the number and the full header in a separate file. Usually, since numbers are much smaller than the full header, this can save a lot of space in terms of both RAM and hard-drive space; the drawback, of course, is that the headers need to be retrieved instead of being available to view directly, but this can be easily remedied with some simple scripts. The benefit of converting headers into numbers becomes much more apparent at larger network sizes.

    b. Often, a data set of more than 1000 sequences will need to be further condensed by CD-hit to make the number of all-by-all BLAST results into a reasonable size. In my personal experience, the largest network that I have processed was a starting set of 90,000 sequences that generated approximately 90 million results (edges) in an all-by-all blast with 'max_target_seq' set to 1000. To decide on the size of your dataset, try conducting CD-hit at various thresholds (e.g., 90%, 70%, 50%) and seeing which dataset size suits your needs.

5. With the final processed data set, conduct an all-by-all BLAST.

4.2 Guide on SSN construction

This section will demonstrate the actual steps involved in generating an SSN as discussed in the workflow.

4.2.1 Data collection

For the example, the data set seeks to explore the serine-beta-lactamase (SBL) superfamily. The data can be separated into two categories based on their purpose. The bulk of the data primarily comprises a set of genomic protein sequences from the UniProt data base, which are mostly uncharacterized sequences but serve to provide a diverse set of protein sequences. In contrast, there are a relatively small number of known (characterized or studied in some way) SBL sequences that degrade beta-lactam antibiotics, and a number of known penicillin binding proteins (PBP) that do not degrade beta-lactam antibiotics but are the hypothesized evolutionary ancestors for most SBLs. The data set with proteins of known function and or evolutionary origin are used to provide hints on the behavior of sequences that are similar to them in the SSN. Thus, it is important to distinguish data sets that have desirable known attributes and keep them as a separate data set (either as a separate file or in a list of indentifiers) from the bulk of data with unknown characteristics, as this will make annotations in the downstream process much easier.

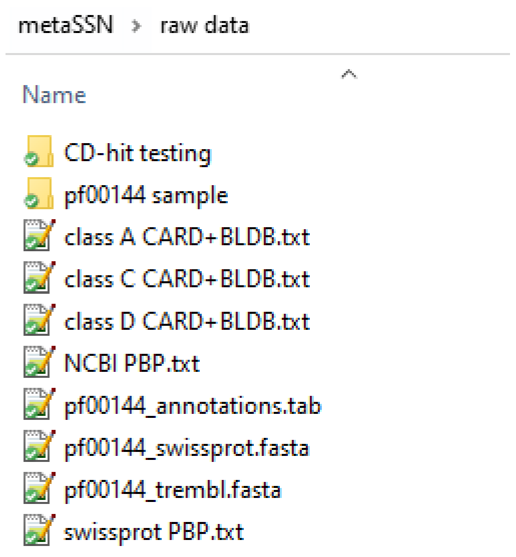
Specifically, the data in the example network comprises subsets (a representative proportion) of data from 4 main sources. The known SBLs, separated into the class A, C and D SBLs, are extracted from the comprehensive antibiotic resistance database (CARD) and the beta-lactamase database (BLDB). The known PBPs are extracted from NCBI and SwissProt (subset of UniProt) based on those identified in a publication on PBPs. The rest of the data are members of the pfam00144 (beta-lactamase) family, which were extracted from Uniprot, along with a table containing various metadata; the SwissProt members of pfam00144 were kept as a separate file from the TrEmbl members. See the 'raw data' folder for each of these respective files. A summary of the dataset to be used is in **Table 1**.

Table 1. Raw data sets

Data set	Status [a]	Source	Sequences	Filename
class A	known	CARD, BLDB	156	class A CARD+BLDB.txt
class C	known	CARD, BLDB	61	class C CARD+BLDB.txt
class D	known	CARD, BLDB	452	class D CARD+BLDB.txt
PBP	known	NCBI	49	NCBI PBP.txt
PBP	known	UniProt (SwissProt)	41	swissprot PBP.txt
pf00144	known	UniProt (SwissProt)	96	pf00144_swissprot.fasta
pf00144	uncharacterized	Uniprot (TrEmbl)	158895	pf00144_trembl.fasta
total			159750	

a) Known sequences mean that the sequences were directly or indirectly associated with the family of proteins in question. The degree of evidence needed to be considered a confirmed representative of the family depends on the source database.

The files in the raw data directory look like the following.


metaSSN > raw data

Name

CD-hit testing
pf00144 sample
class A CARD+BLDB.txt
class C CARD+BLDB.txt
class D CARD+BLDB.txt
NCBI PBP.txt
pf00144_annotations.tab
pf00144_swissprot.fasta
pf00144_trembl.fasta
swissprot PBP.txt

For the purpose of the tutorial, the pf00144 data set from TrEmbl was further reduced to 20% of its size by random sampling using the 'sample_fasta.py' script.

```
python   sample_fasta.py   pf00144_trembl.fasta   30000
pf00144_tr_sample.fasta
```

The sampled sequences are stored in the 'pf00144 sample' folder. The sampling is a random process, so the user should simply use the data set provided as part of the tutorial to ensure they are using the exact same dataset.

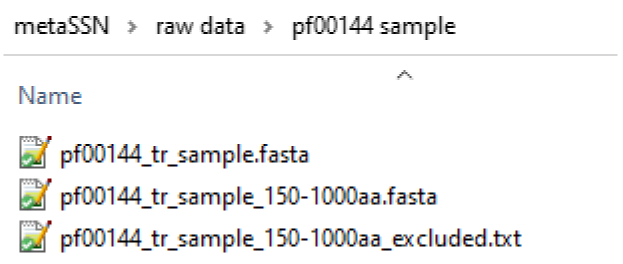
This reduction in sequence count was tested to be appropriate by a sequential CD-hit of 70% followed by 50%, which reduced the 30000 sequences to 14990 clusters at 70% and 7745 clusters at 50%. These results are in the 'CD-hit testing' folder. The tutorial aims to conduct BLAST on a data set clustered at 50%, both to keep the data size small, but also to demonstrate the declustering function of the metaSSN script. For all subsequent steps, the 'pf00144_tr_sample.fasta' will be used in place of the full pf00144 data set.

4.2.2 Data curation

The data curation step is meant to isolate only the data we wish to use for the SSN. By examining the pf00144 family on Uniprot, it can be seen that some sequences in the data set can be as long as 8000 amino acids in length. The typical SBL is around 300 amino acids and the typical PBP is around 800 amino acids. It is up to the user whether they wish to keep extremely long sequences, and in this case we will discard them. To filter for only sequences within a range of lengths, we use the 'filter_fasta_length.py' script as follows.

```
python   filter_fasta_length.py   pf00144_tr_sample.fasta   150   1000
pf00144_tr_sample
```

The results can be examined within the subfolder of 'raw data' called 'pf00144 sample'. There are two files output, one called 'pf00144_tr_sample_150-1000aa.fasta' which contains the sequences that pass the filter, and are thus for use in the SSN. For additional information, the headers of the sequences that were excluded, along with their sequence length, are outputted to the file ‘pf00144_tr_sample_150-1000aa_excluded.txt’. The outputs can be seen as shown below.

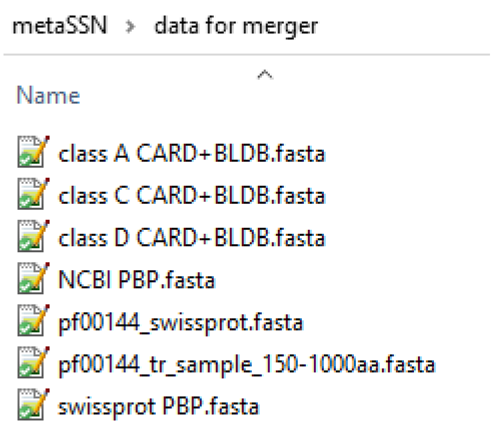

metaSSN > raw data > pf00144 sample

Name

pf00144_tr_sample.fasta
pf00144_tr_sample_150-1000aa.fasta
pf00144_tr_sample_150-1000aa_excluded.txt

If any of the data sets are known to be highly redundant, the user should also perform some sort of redundancy reduction, such as through a CD-hit clustering at 100% identity cut-off. This requirement would apply to data from databases without redundancy filters, such as sequences directly retrieved from NCBI or a metagenomics search. The UniProt database already has a form of redundancy reduction built in, and thus we will skip this step in this tutorial. Keep in mind that the length filter always needs to be done before the CD-hit, since removing sequences after a CD-hit removes an entire cluster rather than just a single sequence.

To prepare the data for merger into the BLAST database, we move all the files into the ‘data for merger’ folder. All the data sets of known data sets (**Table 1**) are moved directly to this folder, along with the length filtered data for the pf00144 TrEmbl data set (pf00144_tr_sample_150-1000aa.fasta). The assembled data in the folder look like the following, with files renamed to the ‘.fasta’ extension for downstream steps.


metaSSN > data for merger

Name

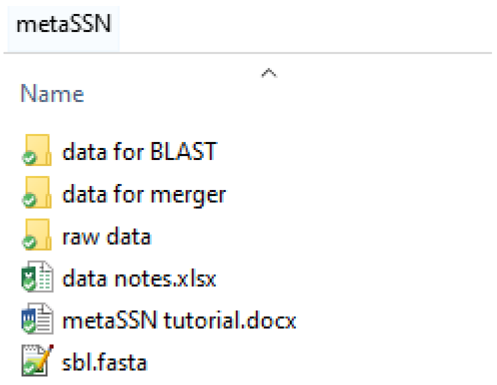
class A CARD+BLDB.fasta
class C CARD+BLDB.fasta
class D CARD+BLDB.fasta
NCBI PBP.fasta
pf00144_swissprot.fasta
pf00144_tr_sample_150-1000aa.fasta
swissprot PBP.fasta

4.2.3 Merging data

The data can be merged into a single file using the ‘merge_fastas_in_directory.py’ script, as follows. The script is run from within the directory ‘data for merger’.

```
python  merge_fastas_in_directory.py   .   ../sbl.fasta
```

For this script to work, all the files inside 'data for merger' must end in '.fasta' or else the script will ignore the file (see **Table 2** for file names). The file creates the fasta file 'sbl.fasta' in the directory just outside 'data for merger'. As shown here:



The merged file contained 3 fewer sequences than expected (**Table 2**). One suspicion was that different files may have the same sequence, which was confirmed not to be the case with the following bash command (also run from the 'data for merger' directory). The script compares the headers between all pairs of fasta files and in the case that an overlap is found, the fasta header starting with a '>' will be printed, otherwise the two files being compared will be the only output. This did not find any repeated sequence headers between files.

```
for f1 in *; do
   for f2 in *; do
      echo "$f1" "$f2";
      if [ "$f1" != "$f2" ]; then
         awk '(NR==FNR && /^>/){a[$0];next} (/^>/ && $0 in a){print $0}'
         "$f1" "$f2";
      fi;
   done;
done
```

It turns out that the repeated sequences were within the same file (1 in class A CARD+BLDB.fasta, and 2 in NCBI PBP.fasta) by checking each file with the following bash script (run from the 'data for merger' directory) that checks for repeated headers and prints them.

```
awk  '/^>/{if($0 in a) print $0;else a[$0]}'  'NCBI PBP.fasta'
```

Table 2. Files for merging into main database

Data set	Status	Source	Sequences	Filename
class A [a]	known	CARD, BLDB	156	class A CARD+BLDB.fasta
class C	known	CARD, BLDB	61	class C CARD+BLDB.fasta
class D	known	CARD, BLDB	452	class D CARD+BLDB.fasta
PBP [a]	known	NCBI	49	NCBI PBP.fasta

PBP	known	UniProt (SwissProt)	41	swissprot PBP.fasta
pf00144	known	UniProt (SwissProt)	96	pf00144_swissprot.fasta
pf00144, sampled and filtered	uncharacterized	Uniprot (TrEmbl)	28983	pf00144_tr_sample_150-1000aa.fasta
total expected			29838	
total observed			29835	

a) These files contained duplicate sequences. Class A contained a duplicate of 1 sequence, while the NCBI file contained duplicates of 2 sequences.

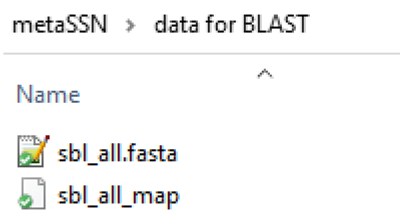
4.2.4 Data preprocessing

An additional modification one can make to the data is changing the fasta sequence headers to numbers. This should be done both before the BLAST and the CD-hit. The benefit of the process is that the numerical identifiers will save a lot of space is all downstream output, compared to the original fasta headers. Numerical identifiers also avoid the issue of headers with spaces in them, which are truncated after the first space by BLAST and CD-hit, leading to additional work required to match sequences in the SSN and the starting sequence data for extraction or annotation. The downside of the numerical identifiers is that they are no longer human readable and would take an additional step to retrieve the original header, which may contain information on the gene name and organism of origin; this is not a major inconvenience if scripts are set up to perform this task, as will be discussed below. If the user believes that file size will not be an issue and that the only problem may be the presence of spaces in the sequence headers, they can simply replace all the spaces with another character, such as the underscore '_'.

In the tutorial, we will convert the merged data 'sbl.fasta' into a file with numerical headers using the following script. Run the script in the same directory as 'sbl.fasta'.

```
python   index_fasta_headers.py   sbl.fasta   sbl_all
```

Two new files are made in the same directory, which are 'sbl_all.fasta' that contains the fasta sequences with headers exchanged by numbers, and 'sbl_all_map' which tracks the mapping of the numbers to the full sequence headers. The files are then moved to the 'data for BLAST' folder, which now look like the following. The user can do a quick check to make sure the headers and mappings still correspond to the correct sequences using any text editor.



Before conducting an all by all BLAST, the user needs to consider he total number of sequences being used for BLAST and reduce the number of sequences through CD-hit clustering if needed. The maximum number of possible results will be the number of sequences in the data set multiplied by the max number of BLAST hits, which we will set to 1000 for the tutorial. This means we can expect ~29,000 × 1000 ≈ 29,000,000 hits in the worst case, which will need to be further processed into the SSN. While this is a manageable number of hits, it is very inefficient to keep so many results for such a small starting data set; with current sequence data sets, it is common to have more than 400,000 sequences without clustering. We will reduce the number of sequences using a sequential CD-hit, first clustering the raw data set to a 90% threshold, then using the 90% cluster representatives to perform another clustering at 70% and finally using the 70% output for a 50% CD-hit. This can be performed using the CD-hit webserver (http://weizhong-lab.ucsd.edu/cdhit-web-server/cgi-bin/index.cgi), but I will also list the commands to run CD-hit using a local installation. The commands are run within the 'data for BLAST' folder, with the file 'sbl_all.fasta' inside of it, using the windows Ubuntu subsystem (or linux/mac terminal).

```
cd-hit    -n 5    -c 0.9    -T 0    -i sbl_all.fasta    -o sbl_90

cd-hit    -n 5    -c 0.7    -T 0    -i sbl_90           -o sbl_70

cd-hit    -n 3    -c 0.5    -T 0    -i sbl_70           -o sbl_50
```

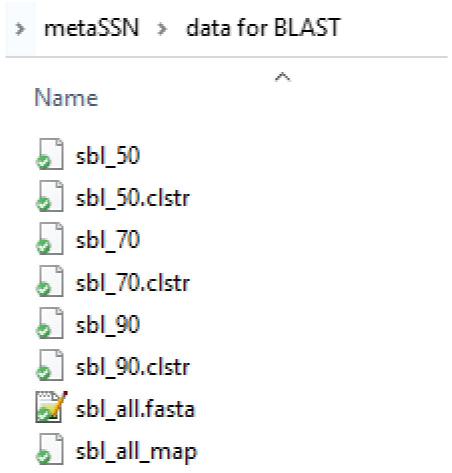
This will generate 3 sets of files, each set representing the results of clustering from all sequences at 90% (sbl_90), clustering from 90% to 70% (sbl_70) and clustering from 70% to 50% (sbl_50). Each set is also comprised of 2 separate files, with one file containing the cluster representatives as a fasta file (no file extension) and the other file containing information on the membership within each cluster ('.clstr' file extension). The number of sequences (cluster representatives) after each step can be seen in **Table 4**.

One approach in making the SSN is to only make use of the representative sequences after CD-hit in a network and other sequences are ignored. However, we aim to extract the other sequences clustered under each representative after the BLAST process and network generation. To do this, the MetaSSN script will simply parse the CD-hit cluster file to see which other sequences are clustered under each representative, provided the CD-hit cluster information contains the exact same labels or identifiers as the BLAST results. Note that by default, CD-hit only keeps part of the fasta header in the cluster information, with a default of 20 characters (not sure if the same restriction applies in the webserver). This limit can be increased to a higher number using the '-d' option. Regardless, the user should ensure that the cd-hit cluster information is still enough to uniquely identify each sequence so that the representatives can be later 'declustered' to retrieve the unclustered sequences. For example, even if the user is confident that truncating the sequence at the first identifier will still provide unique identifiers (which it usually should, if the sequence is from UniProt or NCBI and thus have a standard header format with the accession before the first space), a local run of CD-hit would still need to be configured to make sure all characters in the header before the first space is captured (the option '-d' set to 0 will take the header before the 1st space). In contrast, if the online CD-hit server is used and the spaces are replaced with underscores, the sequence names in the CD-hit file would no longer match those in the BLAST exactly, and additional conversions (before or after the BLAST process) will need to be performed to extract the sequences in each CD-hit cluster. In the tutorial, we use numbers in place of the usual headers and the number of raw sequences is no more than 30000, requiring 6 characters or less, and so the default option does not hinder downstream processes as each header will be captured fully.

Table 4. Sequence count after CD-hit at various thresholds

Clustering identity	Sequences
None	29853
90%	21765
70%	14683
50%	7608

With the sequences clustered, the 'data for BLAST' folder should look like the following. The file 'sbl_50' will be the database file used in the all by all BLAST.



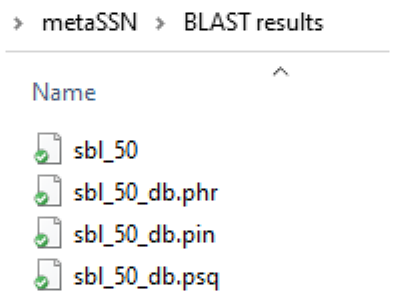
```
metaSSN > data for BLAST
Name
sbl_50
sbl_50.clstr
sbl_70
sbl_70.clstr
sbl_90
sbl_90.clstr
sbl_all.fasta
sbl_all_map
```

4.2.5 All by all BLAST

Finally, to formally construct the SSN, we take the representative data set clustered at 50% identity and perform an all by all BLAST. We first convert the data set to a BLAST database using the following command, inside the 'data for BLAST' folder.

```
makeblastdb  -in sbl_50  -dbtype prot  -out sbl_50_db
```

This generates 3 files with the file prefix 'sbl_50_db' and the suffixes '.pin', '.phr', '.psq'. Move the file into a new folder called 'BLAST results', and also copy the file 'sbl_50' which we will use as the search query. The folder should look like the following.


metaSSN > BLAST results

Name

sbl_50
sbl_50_db.phr
sbl_50_db.pin
sbl_50_db.psq

Now we perform the all by all BLAST using the following command in the 'BLAST results' folder.

```
blastp    -db sbl_50_db    -query sbl_50
-outfmt '6 qacc sacc evalue bitscore'    -out sbl_50_blast_hits
-max_hsps 1    -max_target_seqs 1000    -evalue 1e-5   -num_threads 8
```

Notes:

- '-outfmt' specifies the output format of BLAST. The option demonstrated above tells BLAST to output a table (format 6) and to list the specified columns (qacc – query accession, sacc – subject accession, evalue – E-value, bitscore - bitscore). Keep track of which columns your output format is in. Specifically, the column position of the query, subject and bitscore will be needed for the metaSSN script.
- '-max_hsps' limits the number of hits per subject-query pair to 1. Typically, there can be more than 1 'hsp' or high-scoring segment pair as a result of the BLAST. These could represent different homology regions or partially overlapping homology to different envelopes of the same region. It is up to the user whether they wish to examine more than one 'hsps' in the subsequent SSN.
- '-max_target_seqs' limits the number of hits per query, which is 500 by default. In an SSN, the edge connections are limited by what is included in the blast results. If the 500 hits are all occupied by highly similar sequences (same family), then more interesting connections (between families) may be lost. It is up to the user to decide if the number of hits is sufficient. In my experience, 1000 hits can reach bitscores as low as 50-60 for some queries (though not all, the exact proportions have not been tested), and thus capture more distant connections.
- '-evalue' limits the hits to those with an E-value below the argument to this command. By default the E-value cut-off is 10. It is up to the user to decide what threshold the E-value should be. Here, the E-value cut-off is set to 1e-5 to keep the connections being used for the SSN significant (as opposed to a BLAST search for any potential homologs in a database, which could warrant higher E-values).

This will generate a BLAST results table called 'sbl_50_blast_results', with 6,989,141 separate lines of BLAST results. This is pretty close to the maximum number of hits with the current settings (7,600 × 1000 = 7.6 million), indicating most queries have made use of the maximum number of hits that are still above the significance threshold (below an E-value of 1e-5).

Consider adding instructions for split fasta BLAST. In this case make new python script to split fasta files.

This concludes the steps required strictly for the generation of an SSN. The BLAST results can be loaded directly into any network analysis program (Cytoscape, Gephi, etc.) as a network table. However, the tutorial will proceed to further process the network using the 'MetaSSN' script, to further condense the size of the network and to provide an example of conducting annotations on a clustered network of meta-nodes.

5. Using MetaSSN to condense and annotate an SSN

MetaSSN is a script that performs two major functions in the processing of SSNs. One function is the reduction of network size through clustering of sequences with BLAST bit scores above a certain threshold into a single 'metanode' (node made of nodes). This way, highly similar sequences are grouped together, removing all the non-informative connection information between them; to elaborate, many edges will connect a group of similar sequences, but the connections are not very informative since they are most likely already in the same protein family, where as the most valuable information for an SSN is connection between different families in a superfamily. If the intent is to resolve relationships between individual protein sequences with high resolution, a phylogenetic analysis is much more suitable.

The other major function is to apply a set of user supplied annotations to the SSN. Usually, the annotation of an SSN needs to be done for each network processed at each cut-off, and the user will usually associate the annotations by matching each node in the network with the matching annotations of that sequence (or cluster of sequences). MetaSSN provides a suite of tools to facilitate the matching of metanodes with the annotations of their respective sequences, and is designed so that the annotations only need to be configured once for the network to be analyzed at all thresholds.

5.1 Workflow

1. Collect annotation data
2. Configure MetaSSN
3. Run MetaSSN
4. Post-MetaSSN annotations

5.2 Collecting and curating annotation data

MetaSSN was design to keep a set of annotation files that can be reused for networks analyzed at different cut-offs. Each of these files share a similar structure, which is a simple tab delimited table. In all cases, only 2 columns are read from the table to annotate the network, one column for the sequence of interest and the other for the desired attribute. The user can assemble a full table with multiple columns of different annotations for each sequence, then specify the columns to be used inside the configuration file of MetaSSN. The pattern of a basic annotation file is shown below.

ID	Annotation
0	A
1	B
2	C

With regards to how annotations are assigned to metanodes, the main thing to keep in mind is that a meta-network contains meta-nodes, and that each node can contain multiple sequences from the all by all BLAST (these sequences will be referred to as metanode ‘members’). If the data was also reduced through CD-hit before hand, each hit in the BLAST (or a metanode ‘member’) also corresponds to the list of clustered ‘sequences’. MetaSSN can be used to annotate both at the ‘member’ level, restricting annotations to only sequences directly used in the all by all BLAST, or to annotate all sequences at the ‘sequence’ level. To do this, the CD-hit cluster files need to be provided to the configuration file (See **Section 5.3.2** below).

The following sections will demonstrate how to generate a set of data for use in each of the annotation methods provided by MetaSSN. These will be compatible with both the ‘member’ and ‘sequence’ level annotations. The presence of extra entries also does not disrupt the process, as sequences that are not found in the network (member or sequence level) are simply ignored. To see how to configure these annotation file for use in MetaSSN, see **Section 5.3.3**. All files generated in this section should be moved to an ‘annotations’ folder.

5.2.1 Count of known sequences

Each metanode can be assigned a count for the number of target sequences that are found within that metanode. For example, the most common usage for this is to count the number of known/characterized sequences in a metanode, which informs the user where a certain family of sequences lies within the network and also provides a hint on the behavior of other uncharacterized sequences within that metanode. Another use is to determine if multiple different families are in the same metanode at a certain cut-off. This can be used to inform the current level of separation the user is viewing in the current network and to determine which networks and clustering thresholds should be used for a specific purpose, such as isolating sequences from different families.

As a minor convenience, the visual annotation of the network can also be aided by the annotation in the form of counts. Most network viewing programs (Cytoscape, Gephi) have easy ways to filter for numbers, and the user can easily identify all metanodes that contain any of a set of target sequences. In comparison, if such a search were to be conducted using the sequence name or identifiers in the set of target sequences, then all such identifiers would need to be searched one by one, or the network needs to be processed by script before being loaded into the network viewing program (which is essentially what MetaSSN is doing to make the clusters easy to search for).

To generate a file compatible with MetaSSNs counting function, follow these steps. We will take each of the fasta files with known sequences (class A/C/D SBL or PBP) and list every header side by side with the index of that header in the network. The following illustrates this with the class A SBLs, running the command from within the ‘data for merger’ folder as the files inside represent the curated data set before they are merged into the BLAST database.

```
python  get_fasta_headers.py  'class A CARD+BLDB.fasta'
'classA_headers'
```

This script extracts all the fasta headers into a file called 'classA_headers', with each header occupying a separate line. The following command is then used to convert these headers to the numerical indices used in the network.

```
python   convert_mapping.py  'classA_headers'
'..\data for BLAST\sbl_all_map'  'classA_indices'  -i 1  -v 0
```

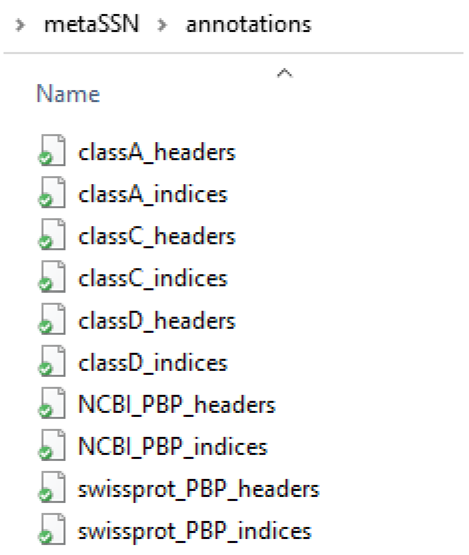
The script takes the headers in 'classA_headers' and extracts the corresponding index inside the mapping file 'sbl_all_map' that was generated earlier in the tutorial which is inside the 'data for BLAST' folder. For each header, the corresponding index is extracted from the mapping file and both header and index are written side by side separated by a tab in the file 'classA_indices'. The file has the following pattern.

```
Header1    index1

Header2    index2

Header3    index3
```

As noted above, this format is suitable for MetaSSN annotations. Repeat this process for the other files of known sequences, the recommended output file names are listed below:

Fasta	Headers	Indices
class A CARD+BLDB.fasta	classA_headers	classA_indices
class C CARD+BLDB.fasta	classC_headers	classC_indices
class D CARD+BLDB.fasta	classD_headers	classD_headers
NCBI PBP.fasta	NCBI_PBP_headers	NCBI_PBP_indices
swissprot PBP.fasta	swissprot_PBP_headers	swissprot_PBP_indices

Move the header and indices files into the 'annotations' folder for organization. See the example image below to see what the folder looks like.



classA_headers
classA_indices
classC_headers
classC_indices
classD_headers
classD_indices
NCBI_PBP_headers
NCBI_PBP_indices
swissprot_PBP_headers
swissprot_PBP_indices

5.2.2 Label of known sequences

The ‘label’ annotation option takes a list of labels for a set of sequences, searches through each metanode to and applies the label to the metanode if a sequence with a label is found. This feature was designed to allow the user to easily pick out specific sequences. It complements the ‘count’ annotation option, since the ‘count’ annotations are easy to search but do not provide information on specific sequences. For cleanliness, when multiple of the same labels are encountered, the label is only listed once.

To generate a set of labels for the tutorial, we can take the ‘indices’ file from the ‘count’ annotations, as it already presents a mapping from the original sequence header to the sequence index in the network. The generation of labels relies mostly on user judgement and is not always easy to automate, since it is dependent on the user’s needs. For example, if we have two sequences from closely related families (e.g. AMP-type and CTX-M-type sequences from the class A SBLs), the user can choose to name each sequence within its own family (AMP or CTX-M), or label them based on the shared larger family (class A SBL).

In this tutorial, we will take the class A sequences from ‘classA_indices’ and generate the labels in Excel. Copy the contents of ‘classA_indices’ into an excel file called ‘label_processing.xlsx’ inside of the ‘annotations’ folder. In the 3rd column, start typing the name of the gene without the variant number.

Example header:

```
gi|5596421|emb|CAB51471.1|ACI-1| beta-lactamase [Acidaminococcus fermentans]
```

The gene label we are interested in is ‘ACI-1’, but we will only take the label of the gene and exclude the variant number. This will make it so that multiple gene variants that only differ by a small number of mutations are labelled as the same group. The output in the Excel should look something like the following:

```
gi|5596421|emb|CAB51471.1|ACI-1| beta-lactamase [Acidaminococcus fermentans]        0    ACI
```

gi|2992469|gb|AAC09015.1|AER-1| AER-1 precursor [Aeromonas hydrophila]	1	AER
gi|40950501|gb|AAR97884.1|AFA-1| AmpC [Aliivibrio fischeri]	2	AFA
gi|4493255|emb|CAA37699.1|AMA-1| beta-lactamase [Actinomadura sp. R39]	3	AMA

As a minor convienience, the formatting of most UniProt and NCBI sequences headers may also be compatible with the 'Flashfill' option in Excel. If the user cannot get 'Flashfill' to recognize the pattern then extracting the gene name by script is also possible. Regardless, the user needs to make specific decisions on what a sequence's label should be.

To make this a file compatible with MetaSSN, simply copy and paste the Excel sheet contents into a new file and name it 'classA_labels'. Move this file to the 'annotations' folder.

5.2.3 Frequency of host taxonomies

The 'frequency' annotation aims to provide frequency summaries of categorical data within each metanode. For example, we can assign a host organism to most UniProt or NCBI sequences and use the 'frequency' annotation to calculate the fraction belonging to a certain species. In terms of network viewing, the 'frequency' annotation is not necessarily good for searching, but it provides valuable information on metanode composition. Keep in mind that like any other summary based on categorical data, having too many categories will make the frequencies harder to interpret, such as if the 'frequency' annotation is used on something like 50 different categories. In the case of too many categories, the user can condense them first using their own understanding of the data. However, if the goal is check for a certain dominant category, then the number of categories is not an issue as the dominant category is the only one of interest.

To create a file of annotations, we will make use of the annotations table downloaded from UniProt ('pf00144_annotations.tab' in the 'raw data' folder). Because the table is already a tab-separated table file, it can be used directly for MetaSSN annotations, we simply need to match each entry to its associated row in the table.

To start, we will repeat the steps in the 'count' annotation section, and generate a file that links the original headers of the pf00144 sequences with its sequence index in the network. Run the following commands in the 'data for merger' folder:

```
python   get_fasta_headers.py   'pf00144_tr_sample_150-1000aa.fasta'
'pf00144_trembl_headers'

python  convert_mapping.py  'pf00144_trembl_headers'
'..\data for BLAST\sbl_all_map'  'pf00144_trembl_indices'  -i 1  -v 0
```

Take the 'pf00144_trembl_indices' and use the following script inside the 'data for merger' folder to extract the UniProt accessions:

```
python   uniprot_acc_extract_fasta.py   'pf00144_trembl_indices'
'pf00144_trembl_acc'
```

The output ‘pf00144_trembl_acc’ now contains an extra column compared to the starting file, which lists just the UniProt accession of each squence. We will add some column headers to this file. Open the file and manually type ‘header’, ‘index’ and ‘accession’ as the first line separated by the tab character. Then run the following command, still inside the ‘data for merger’ folder:

```
python   merge_tables_by_column.py  'pf00144_trembl_acc'
'../raw data/pf00144_annotations.tab'   'pf00144_table'   -id_col1 2
-id_col2 0   -h2   -h1
```

This script merges the ‘pf00144_trembl_acc’ table with the UniProt table ‘pf00144_annotations.tab’ inside the ‘raw data’ folder. The merger is conducted based on the Uniprot accession, which is column 3 in the accession file (index 2) and column 1 in the UniProt table (index 0). This output allows MetaSSN ‘frequency’ annotation option to be used to summarize the frequencies of host organisms at various levels; metadata for a sequence’s ‘Genus’, ‘Class’ and ‘Superkingdom’ will be illustrated during the tutorial, but the user can obtain more from UniProt by adding columns to the search results.

Move this file to the ‘annotations’ folder.

5.2.4 Distribution of sequence lengths

One other piece of information that may be of interest is the length of the sequences within a metanode. Differences in nodes could demarcate different families due to the addition or loss of extra domains. The ‘distribution’ annotation option takes a list of numerical data and produces a 5 number summary in the following format: min-Q1-Median-Q3-max. If there are 5 or less than sequences, the lengths are simply listed.

This annotation method seeks to give a rough overview of a distribution of values. For a higher resolution distribution (such as a histogram), the user would need to extract the sequences from the metanode and then plot the distribution.

Extraction of sequence lengths is relatively simple, since the information is contained in the sequence itself. Use the following command on the ‘sbl_all.fasta’ in the ‘data for BLAST’ folder.

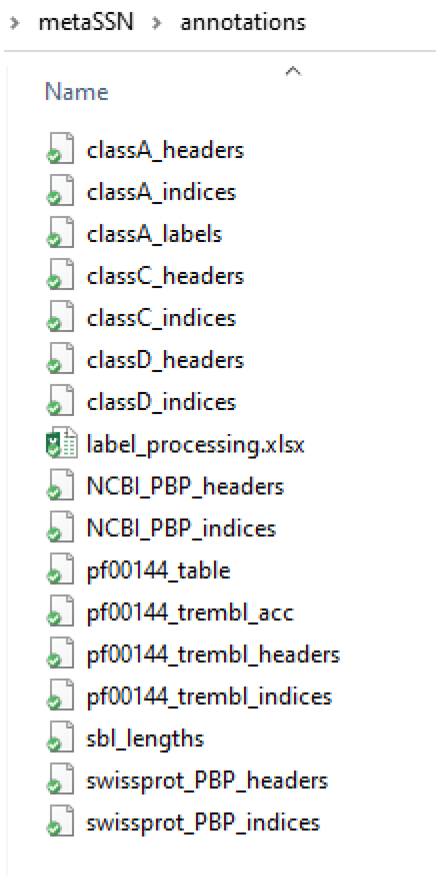
```
python  get_fasta_lengths.py   sbl_all.fasta   sbl_lengths
```

This will create a new file called ‘sbl_lengths’, which is a file that lists the header and the length of the sequence belonging to that header. Since we used the command directly on the fasta with headers converted to indices, the output file will be directly compatible with the MetaSSN annotations.

Move this file to the ‘annotations’ folder.

5.2.5 Final overview

After following the above listed steps, the ‘annotations’ folder should look like the following:



metaSSN › annotations

Name

classA_headers
classA_indices
classA_labels
classC_headers
classC_indices
classD_headers
classD_indices
label_processing.xlsx
NCBI_PBP_headers
NCBI_PBP_indices
pf00144_table
pf00144_trembl_acc
pf00144_trembl_headers
pf00144_trembl_indices
sbl_lengths
swissprot_PBP_headers
swissprot_PBP_indices

5.3 Configuring MetaSSN

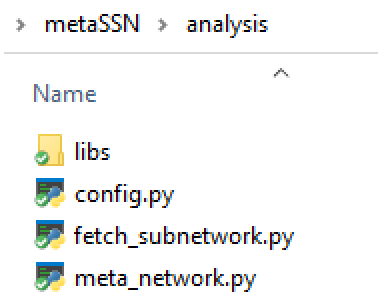
This section will show the user how to configure the MetaSSN script to produce an annotated SSN from the data generated in the previous steps. The configuration file is written in native Python code, and the user needs a bit of knowledge on the different Python data types. Furthermore, the user needs to use some sort of text editor with syntax highlighting for python (Notepad++ or Sublime Text) or coding software to allow for coloring of the code (it will make using the file way easier). Keep in mind Python counting starts at 0, so for options where the user wishes to indicate a position the user should subtract 1 from the position from common place numbering (the 1st column in a table file is the 0th column in Python).

The tutorial will make use of most options in the MetaSSN script, but not all of the options. To see a list of all options and their purpose, see the MetaSSN manual.

5.3.1 Environment setup

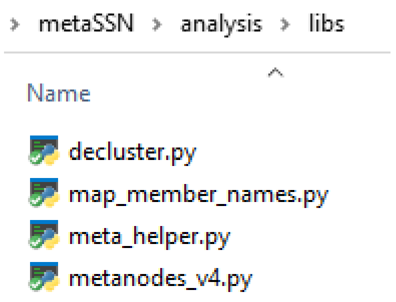
First, we will set up an environment from which the MetaSSN script will operate. The MetaSSN scripts are very small, and can simply be copied into the most convenient folder for analysis.

Move the contents of the ‘MetaSSN_v4’ folder into a folder called ‘analysis’, there should be 3 python files (‘config.py’,’meta_network.py’, and ’fetch_subnetwork.py’) plus a folder called ‘lib’. The folder should look like the following.



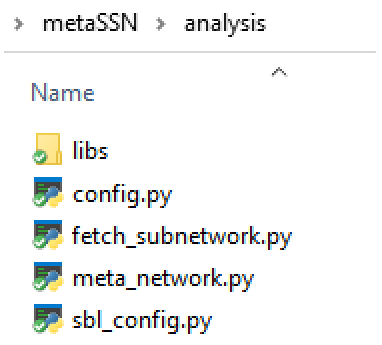
```
› metaSSN › analysis
Name
libs
config.py
fetch_subnetwork.py
meta_network.py
```

In side, of libs are 4 helper scripts. The user does not need to interact with them, but they should check that the following 4 files are present.



```
› metaSSN › analysis › libs
Name
decluster.py
map_member_names.py
meta_helper.py
metanodes_v4.py
```

To configure the MetaSSN script, we will make a copy of the configuration file. Inside the ‘analysis’ folder, copy the ‘config.py’ file and rename it to ‘sbl_config.py’, like so:



```
› metaSSN › analysis
Name
libs
config.py
fetch_subnetwork.py
meta_network.py
sbl_config.py
```

Open up the ‘sbl_config.py’ file in a text editor before continuing to the following sections.

5.3.2 BLAST result input and MetaSSN output

The main function of MetaSSN is to take a set of blast results in tabular format, and process them into a sequence similarity network. The user specifies the input BLAST results using first 2 options, ‘blast_results_path’ and ‘blast_results_files’.

‘blast_results_path’ is the path to the folder containing the blast results, specified as a python string. In this case we will set it to the following:

```
blast_results_path = '../BLAST results/'
```

‘blast_results_files’ is a python list of file names (each a string) within the ‘blast_results_path’ that should be used in the generation of the SSN. We will set it to the following.

```
blast_results_files = ['sbl_50_blast_hits']
```

Following these settings, the user also needs to inform MetaSSN which columns of data in the BLAST results correspond to the BLAST query, subject and bitscore. If the user followed the tutorial or are using the same BLAST results, the queries are in the 1st column, the subjects in the 2nd and the bitscore in the 4th. The settings for MetaSSN will then look like the following:

```
query_col = 0
subject_col = 1
bitscore_col = 3
```

Finally, the user needs to specify the location and name of the output networks using the ‘output_label’ and ‘output_dir’ settings. The following settings are used:

```
output_label = 'sbl_50'
output_dir = 'sbl_SSN/'
```

This will instruct MetaSSN to save the output in a new folder called ‘sbl_SSN’, created in the same folder as the MetaSSN script. Each set of network files in the output folder will be labelled with the prefix ‘sbl_50’.

5.3.3 CD-hit cluster files

MetaSSN can take our representative sequences at 50% CD-hit clustering and expand them to all starting sequences, allowing each metanode to be represented by both sequences directly used in the BLAST (metanode ‘members’) or as all sequences in the starting data set (metanode ‘sequences’).

To do this, we will supply 3 settings to the configuration file. The ‘cluster_info_path’ setting is used to indicate the folder where the CD-hit cluster files are stored. In our case, this would be the ‘data for BLAST’ folder. The setting is as follows:

```
cluster_info_path = '../data for BLAST/'
```

Next, we specify the file within the folder that directly tracks clustering of the representative sequences used in the all by all BLAST. For this we supply the CD-hit clustering conducted at the lowest threshold of 50%, which can then be used to work back through a sequential set of CD-hits to get the starting, unclustered sequence set. The setting ‘cluster_file_info’ is a python dictionary, with the name of the file as the ‘key’ and additional information about that file stored in a ‘value’ (also in the format of a dictionary). The setting looks like the following:

```
cluster_file_info = {

'sbl_50.clstr': {'format':'cdhit', 'delim':None, 'key_col':None,
'member_col':None, 'member_delim':None}

}
```

In this example, 'sbl_50.clstr' is the name of the cluster file within the 'data for BLAST' folder. The additional settings that follow ('format':'cdhit') indicate that this should be treated as a CD-hit cluster file (as opposed to a table, where the cluster members are listed in a column). All other additional settings are set to 'None' since they only apply if the clustering was recorded in a table format.

Finally, to allow MetaSSN to recover all starting sequences processed through a sequential CD-hit, we provide a list of files to the 'cdhit_hierarchy' dictionary in the following fashion:

```
cdhit_heirarchy = {'sbl_50.clstr': ['sbl_70.clstr','sbl_90.clstr'] }
```

The setting is provided as a dictionary, where each 'key' should correspond to a file that was listed in 'cluster_file_info', in this case, 'sbl_50.clstr'. The 'value' of each 'key' in the dictionary should be a list of cluster files of increasing clustering thresholds.

5.3.4 Annotation files

We will also configure MetaSSN to automatically apply annotations to the metanodes in the networks it generates. We first indicate where the annotations are found, by providing a path to the folder containing the annotations to the 'annot_path' setting.

```
annot_path = '../annotations/'
```

Next, we will configure the annotation files for each annotation type. As noted before, the inputs for annotations follows a simple tabular format, with the sequence ID in one column and the desired annotation in another column. For this reason, all the annotation options share the same input format, which is used to describe the input table to MetaSSN. The first entry in the 'count' annotation will be used to illustrate these settings:

```
membership_count_annot = [

{'files':'classA_indices',  'id_col':1,   'data_col':None, 'delim':'\t',
'label':'classA_SBL_count', 'level':'sequence'}

                          ]
```

Each set of annotations is a python list, where each item in the list is a dictionary with additional settings. In each dictionary, the 'files' option can be a single file (string) or a list of files (list of strings) indicating the file with the annotations. The 'id_col' and 'data_col' settings are used to indicate which column in the table should be used to identify sequences and which column contains the data. The 'delim' setting tells MetaSSN how to separate different columns in the table, which for all files in the tutorial is the TAB character. The 'label' setting tells MetaSSN how to label this annotation in the nodes table of the network. Finally, the 'level' option tells MetaSSN at what level to process annotations, which could be 'member' (only sequences directly involved in the all by all BLAST) or 'sequence' (all sequences in the starting data set, only applicable if the CD-hit declustering options have been used).

Following this pattern, we will fill in the count annotations for the 'membership_count_annot' setting:

```
membership_count_annot = [

{'files':'classA_indices','id_col':1,'data_col':None,'delim':'\t','label':
'classA_SBL_count','level':'sequence'},

{'files':'classC_indices','id_col':1,'data_col':None,'delim':'\t','label':
'classC_SBL_count','level':'sequence'},

{'files':'classD_indices','id_col':1,'data_col':None,'delim':'\t','label':
'classD_SBL_count','level':'sequence'},

{'files':['NCBI_PBP_indices','swissprot_PBP_indices'],'id_col':1,'data_col
':None,'delim':'\t','label':'PBP_count','level':'sequence'},

{'files':['classA_indices','classC_indices','classD_indices'],'id_col':1,'
data_col':None,'delim':'\t','label':'SBL_any_count','level':'sequence'}

                         ]
```

Note that for the 4th entry, we combine the PBP annotations by listing them together in the 'files' option. We take advantage of this feature to also combine all the known SBLs into an additional annotation so that they can be easily searched (5th entry), without needing to make a new combined annotations file.

Next are the 'frequency' annotations, supplied to the 'membership_freq_annot' setting:

```
membership_freq_annot = [

{'files':'pf00144_table','id_col':2,'data_col':7,'delim':'\t','label':'gen
us_freq','level':'sequence','highest_entries':5},

{'files':'pf00144_table','id_col':2,'data_col':8,'delim':'\t','label':'sup
erkingdom_freq','level':'sequence','highest_entries':5},

{'files':'pf00144_table','id_col':2,'data_col':9,'delim':'\t','label':'cla
ss_freq','level':'sequence','highest_entries':5}
                        ]
```

We make use of the same table file, 'pf00144_table', which is the UniProt table of annotations on the taxonomic information of each sequence's host organism. The table contains different information in different columns, and that is what we indicate to MetaSSN by supplying the same file multiple times with different 'data_col'. If the user examines the table they will notice we are supplying the 'Genus' (column 8), ‘Superkingdom’ (column 9) and the ‘Class’ (column10). Note that the frequency annotations have an extra option compared to the other annotations, which is the ‘highest_entries’ option. This indicates the maximum number of categories (in order of largest to smallest) to show as frequencies, with the remainder combined into a category called ‘other’.

The following setting is for the ‘label’ annotations, supplied to the ‘membership_label’ setting:

```
membership_label =  [

{'files':'classA_labels','id_col':1,'data_col':2 ,'delim':'\t',
'label':'classA families', 'level':'sequence'}

                    ]
```

Finally, we supply the ‘distribution’ annotation to the ‘membership_dist_annot’ setting:

```
membership_dist_annot = [

{'files':'sbl_lengths','id_col':0,'data_col':1,'delim':'\t','label':'lengt
h_dist','level':'sequence'}

                        ]
```

5.4 Running MetaSSN

5.4.1 Run MetaSSN

Once the configuration file is set up, the user can run MetaSSN using the following command inside the ‘analysis’ folder:

```
python   meta_network.py  100  300  -si 50   -cf sbl_config.py
```

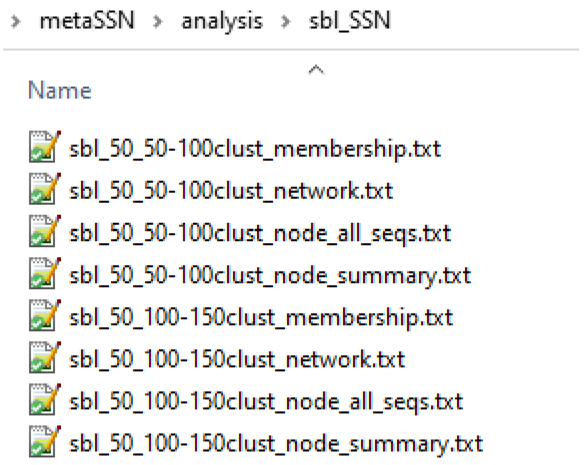
The ‘100’ and ‘300’ tell MetaSSN what the bit score cut-offs are for connecting different metanodes and clustering sequences into the same metanode, respectively. If supplied alone, MetaSSN will generate the network with sequences sharing a BLAST bitscore greater or equal to 300 clustered into the same metanode, and with different metanodes to be connected by edges if any of their members share a BLAST bitscore of at least 100. In addition, the ‘-si 50’ option (stands for ‘sweep_interval’) tells MetaSSN to generate a series of networks at intervals of 50 bitscore between the cluster and edge bitscores. In this case, the sweep starts at 100 cluster bitscore and 50 edge bitscore (50-100) and then proceeds to increasing thresholds until a cluster bitscore of 300 is reached (100-150, 150-200, 200-250, 250-300). Finally, the ‘-cf sbl_config.py’ (starts for ‘config_file’) indicates that we are using the ‘sbl_config.py’ as the configuration for this analysis.

For a list of all available command line options for MetaSSN, consult the manual or examine the help prompt with the following command:

```
python    meta_network.py    -h
```

5.4.2 Examining the output

According to our configurations, MetaSSN will deliver the results to a folder called 'sbl_SSN' within the 'analysis' folder. For each network generated at a set of edge and clustering thresholds, there are 4 files. Each file is prefixed with 'sbl_50' followed by the clustering threshold (e.g. '50-100clust') a suffix that indicates their contents. An example of the output can be seen below.



```
› metaSSN › analysis › sbl_SSN

Name

sbl_50_50-100clust_membership.txt
sbl_50_50-100clust_network.txt
sbl_50_50-100clust_node_all_seqs.txt
sbl_50_50-100clust_node_summary.txt
sbl_50_100-150clust_membership.txt
sbl_50_100-150clust_network.txt
sbl_50_100-150clust_node_all_seqs.txt
sbl_50_100-150clust_node_summary.txt
```

The files with the '_network.txt' suffix are the network files, which are sufficient to produce a visualization in a network viewing software. It has the following contents:

```
node1    node2 bitscore
0        2     181.0
0        4     197.0
0        3     199.0
0        1     184.0
3        4     183.0
7        10    193.0
```

Each row is an edge in the network, between 'node1' and 'node2', with the highest bitscore between connecting sequences listed under 'bitscore'.

The files with '_node_summary.txt' are the node table files, which are used to apply node information and annotations to the network viewing software. It has the following contents:

```
node     member_count      seq_count   sequence:classA_SBL_count     …
0        36                138         135                           …
```

Each row represents a particular metanode, and each column indicates information associated with the metanode. By default, the node table will always have the 'member_count' column, indicating the number of sequences directly observed during BLAST that are clustered in the metanode. If the CD-hit declustering is conducted, the ‘seq_count’ column will indicate the total number of starting sequences in the metanode. Every column afterwards is for annotations supplied by the user.

The above two file types are sufficient to provide the user the ability to view and manipulate the SSNs. MetaSSN also produces two other files for additional information about the sequences observed in each metanode. The ‘_membership.txt’ files list the metanode sequences arising directly from the all by all BLAST, while the ‘_node_all_seqs.txt’ lists metanode sequences arising from the starting data set if a CD-hit declustering is conducted.

5.5 Post-MetaSSN annotations

5.5.1 Updating old networks with new annotations

To update annotations on existing networks, simply update the annotation settings in the configuration file and run the script again with the same settings. By default, MetaSSN will try to use an old network with the same edge and clustering thresholds if those networks are found. Similarly, MetaSSN will try to reuse old declustering information in the ‘_node_all_seqs.txt’ files. The recycling of old networks saves time by excluding the most time consuming steps of the MetaSSN process, allowing the user to easily update annotations. The user needs to keep in mind that recycling can only occur if the output files have the exact same file names, meaning they output folder, output label and edge and clustering bitscore thresholds have to be the exact same as the old analyses.

In the event that the user has acquired new BLAST data, or needs to correct an old network, the user can simply delete the old network files. If the user wishes to be safe and force MetaSSN to reconstruct the networks, they can use the following command line options.

```
python  meta_network.py  100  300  -si 50  -cf sbl_config.py  -R  -RD
```

The ‘-R’ option forces MetaSSN to rebuild the network, and ‘-RD’ forces MetaSSN to redo the declustering step.

5.5.2 Mapping nodes in networks with higher node clustering cut-offs to networks with lower cut-offs

The nature of SSN analysis often involves examining the same network at different thresholds of connection. For example, at lower clustering bitscore thresholds, more divergent sequences may get combined into the same metanode to create a network representing broad family relationships. Taking the same data and clustering at a higher threshold leads to metanodes that represent a more narrow and specific set of sequences (subfamily or subgroup level).

A network can only be viewed at one threshold at a time, but information on behavior from lower thresholds can be assigned to networks at higher thresholds as annotations. This can be used to track the ‘spreading out’ of sequences from a node in which they were initially clustered. For example, a metanode in a network at a low cluster threshold may contain known members of a specific family, while at a higher threshold the metanodes in the network separate the specific members of that family. The annotation to map nodes can be used to generate a network that shows both the individually separated members, but also highlights all metanodes that are part of the family (the same node at the low threshold). Note that the intention of this annotation is to be able to easily color or label this feature in a network viewing software, and only 1 node is listed. This works as intended if labelling a network of higher clustering thresholds with the equivalent nodes from a network of lower clustering threshold, because each lower threshold nodes would be shared by multiple higher threshold nodes (and hence a single number is sufficient). When the opposite occurs, a lower threshold node would correspond to multiple higher threshold nodes, and only one of the nodes would be captured. In short, only use this feature to annotate higher threshold nodes with their membership at lower threshold nodes. The other annotation options will already naturally identify lower threshold nodes as well as higher threshold nodes.

To perform this, we copy the '_membership.txt' files from the network analysis at the '50-100clust' and '100-150clust' levels to the 'annotations' folder. Then we set the following setting in the configuration file.

```
membership_lower_node = [

{'files':'sbl_50_50-100clust_membership.txt','id_col':1,'data_col':0
,'delim':'\t', 'label':'50-100clust node ID', 'level':'member',
'bitscore':100},

{'files':'sbl_50_100-150clust_membership.txt','id_col':1,'data_col':0
,'delim':'\t', 'label':'100-150clust node ID', 'level':'member',
'bitscore':150}

                        ]
```

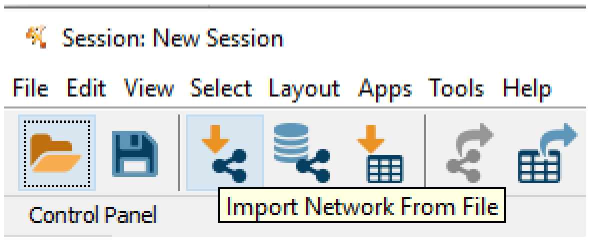
Simply run MetaSSN as before and these annotations should be applied.

6. Network manipulation

6.1 Analysis using Cytoscape

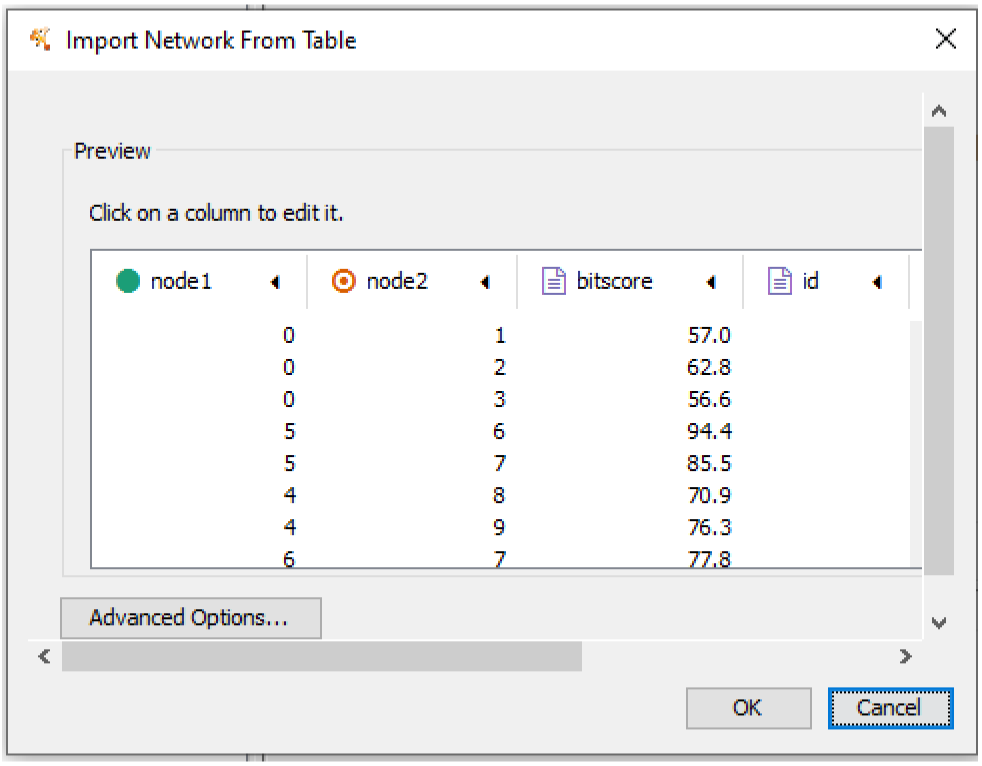
<u>*6.1.1 Loading networks into Cytoscape*</u>

To load a network into a network viewing program, 2 of the output files of MetaSSN need to be used. To generate the network itself, most software will take a network table as input that describes the connections between different nodes in the network. The corresponding files from MetaSSN are the files ending in '_network.txt'. Open Cytoscape and choose 'Import Network From File' in the top tool bar.



Alternatively, you can use the 'File' menu and navigate to Import > Network > File … (short cut key of ctrl+L on Windows).

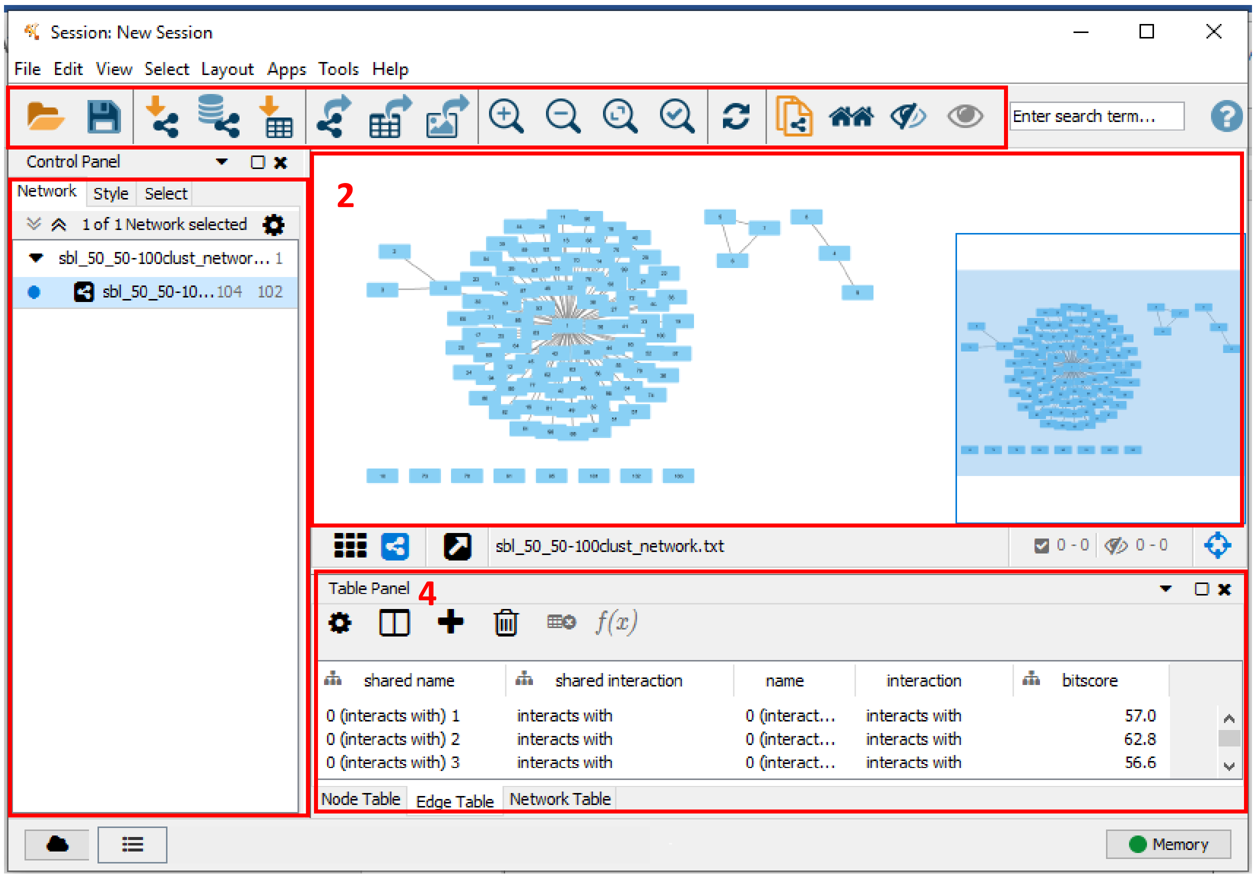
When prompted, navigate to the folder that contains the MetaSSN output, and select one of the network files. The following example will open the network contained within 'sbl_50_50-100clust_network.txt'. This will lead to a dialogue prompting the user for additional information on the network.



node1	node2	bitscore	id
0	1	57.0	
0	2	62.8	
0	3	56.6	
5	6	94.4	
5	7	85.5	
4	8	70.9	
4	9	76.3	
6	7	77.8	

As can be seen, the network file has been parsed into separate columns, with the headers of the columns at the top of the table. MetaSSN generates a network file that can be loaded into Cytoscape without changing any settings. For the sake of elaboration, there are two key elements being considered when loading the network, which are the 'source node' (green filled circle) and the 'target node' (orange dotted circle). Each element corresponds to a column in a table, where the column contains the node ID, such that Cytoscape will construct a connection (edge) between each node in the 'source node' column and the corresponding node in the 'target node' column. The user simply needs to make sure these columns are correctly selected. All other columns are simply annotations for the edges, which can be used for filtering edges if needed.

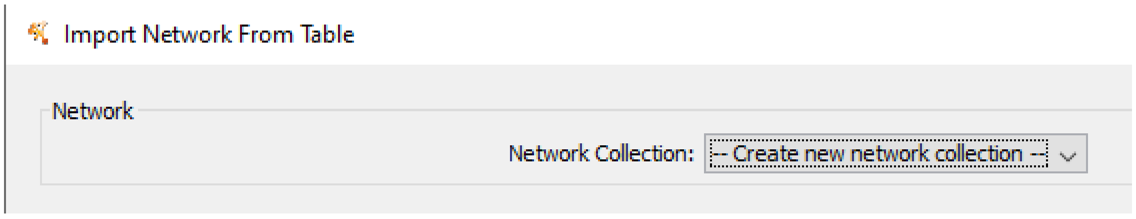
Click 'OK' to load the network, and the user should see something like the following.



shared name	shared interaction	name	interaction	bitscore
0 (interacts with) 1	interacts with	0 (interact...	interacts with	57.0
0 (interacts with) 2	interacts with	0 (interact...	interacts with	62.8
0 (interacts with) 3	interacts with	0 (interact...	interacts with	56.6

The 4 most relevant parts of the Cytoscape interface are highlighted. Panel 1 is the tool bar, which contains some buttons for commonly used operations. Panel 2 is the network view, where the network is displayed. Panel 3 is the control panel, which serves for selecting different networks, setting visualization styles and selecting data within the network. Panel 4 is the Table view, where the network's edge and node tables can be viewed.

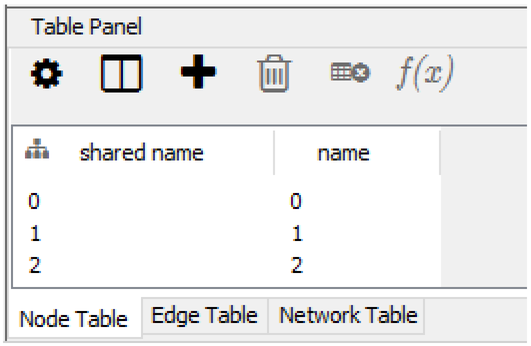
As can be seen, the newly loaded network can be seen in the network view (panel 2). And a corresponding entry is added to the 'Network' tab of the control panel (panel 3). When loading a network into a session with another network open, the 'Network Collection' option also becomes available. This setting tells Cytoscape which 'collection' (entry with triangle in the 'Network' tab of the control panel) to add the network to. Unless the user knows that the node annotations are exactly the same between the networks, the network should always be added to its own collection, as shown below.



Network Collection: -- Create new network collection --

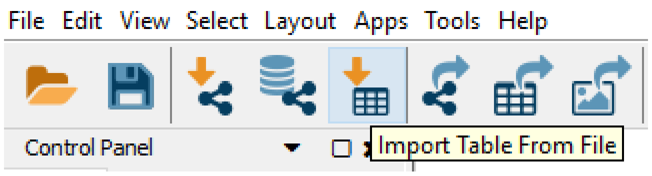
Upon loading a network, Cytoscape will automatically arrange the network using the 'Perfuse force directed layout'. For the sake of SSN analysis, this is the only layout that the user will need, as it clusters nodes together based on the interconnection between nodes. This layout allows for highly interconnected regions to be visually identified in the network as 'clusters' of nodes. To compare different layout options, check the Cytoscape manual or any public material network analysis. If the user needs to reapply the layout (such as when they moved or deleted some nodes) they can find in the top menus under Layout > Perfuse Force Directed Layout > (none). The last option in the layout menu determines what values to use weigh edges (stronger or weaker 'pull') during the clustering step, and unless the user has a specific intent it is recommended to use no weights, hence selecting (none).

If the user navigates to the 'Node Table' tab under the Table Panel (panel 4), they will notice that the nodes are already present, but the annotations are absent.

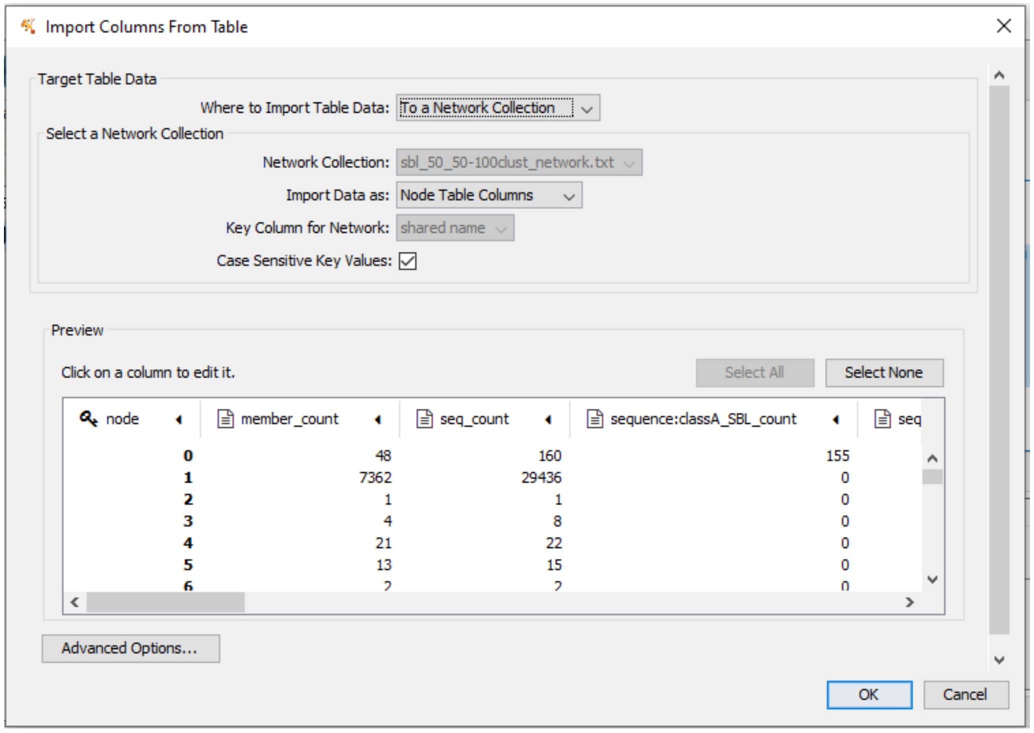


shared name	name
0	0
1	1
2	2

To add annotations to the nodes, we will load in the second file type, the files ending in '_node_summary.txt'. Click the button 'Import Table From File' on the tool bar, or use the 'File' menu and navigate to Import > Table > File…



A file selection menu will appear. Select the '_node_summary.txt' file that corresponds to the same set of network thresholds as the network file ('sbl_50_50-100clust_node_summary.txt' in the case of the tutorial). It is important that the node table always matches the same thresholds as the network, or else the annotations will be wrong. The following menu should appear.



Import Columns From Table

Target Table Data

Where to Import Table Data: To a Network Collection

Select a Network Collection

Network Collection: sbl_50_50-100clust_network.txt

Import Data as: Node Table Columns

Key Column for Network: shared name

Case Sensitive Key Values: ☑

Preview

Click on a column to edit it.

node	member_count	seq_count	sequence:classA_SBL_count	seq
0	48	160	155	
1	7362	29436	0	
2	1	1	0	
3	4	8	0	
4	21	22	0	
5	13	15	0	
6	2	2	0	

The user simply needs to ensure the key column (key symbol) is set to the ‘node’ column. The key column is the information used to match the current nodes in the node table with the newly imported ones. Click OK, then examine the ‘Node Table’ under the ‘Table Panel’ (panel 4) again. The user should now see all the new annotations.

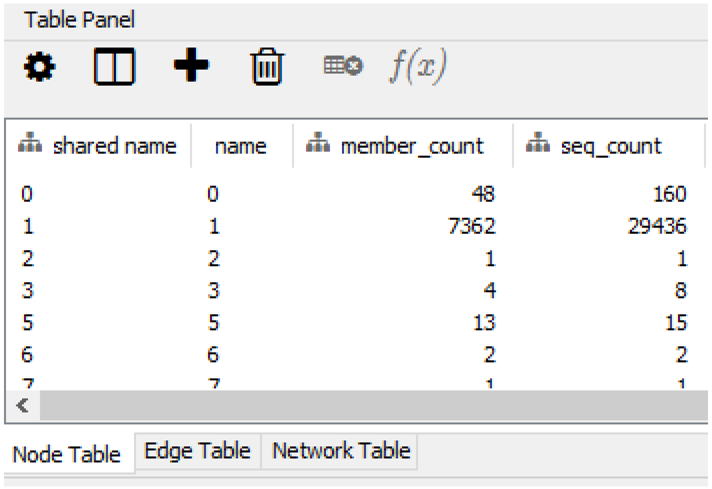
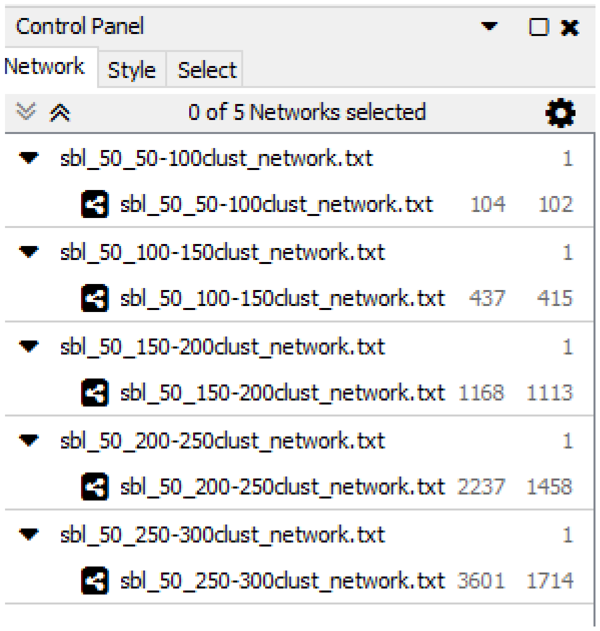


Table Panel

shared name	name	member_count	seq_count
0	0	48	160
1	1	7362	29436
2	2	1	1
3	3	4	8
5	5	13	15
6	6	2	2
7	7	1	1

Load all the network files and their corresponding node tables from MetaSSN into CytoScape. Your ‘control panel’ (panel 3) should look like the following.



sbl_50_50-100clust_network.txt		1
sbl_50_50-100clust_network.txt	104	102
sbl_50_100-150clust_network.txt		1
sbl_50_100-150clust_network.txt	437	415
sbl_50_150-200clust_network.txt		1
sbl_50_150-200clust_network.txt	1168	1113
sbl_50_200-250clust_network.txt		1
sbl_50_200-250clust_network.txt	2237	1458
sbl_50_250-300clust_network.txt		1
sbl_50_250-300clust_network.txt	3601	1714

Once the user has successfully loaded all the networks, save the Cytoscape session.

6.1.2 Selecting nodes

To select a node visually from the network, simply click on the node. The selected node will be highlighted in yellow, and the corresponding entries in the node table will also be selected. As can be seen below.

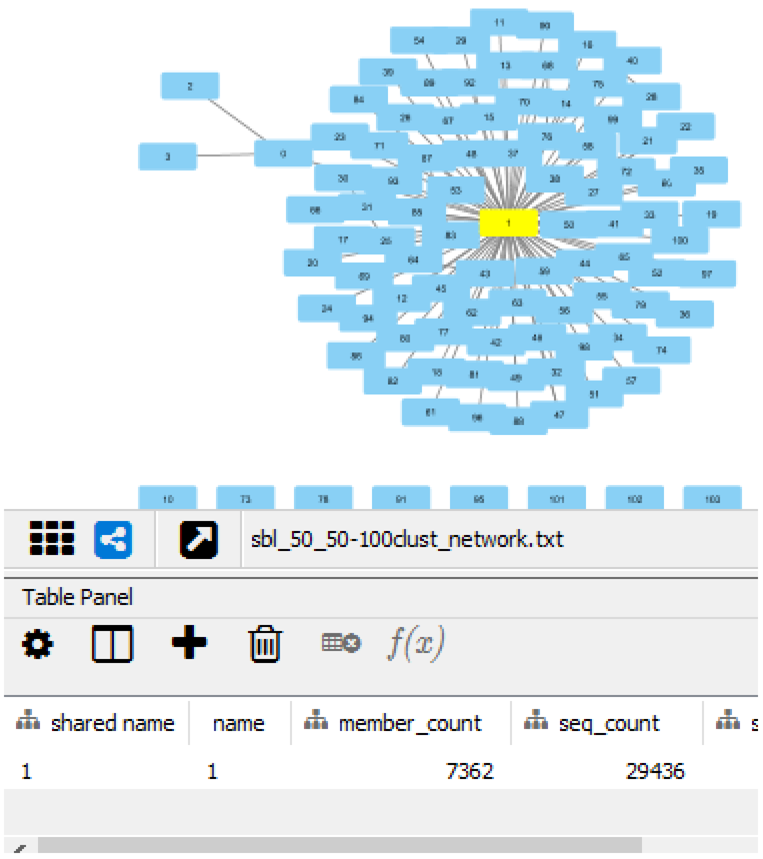
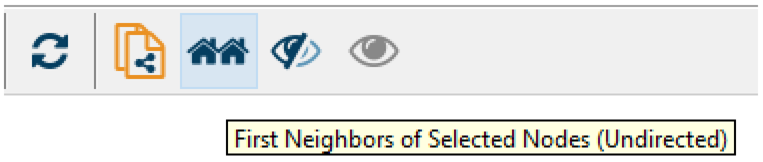

sbl_50_50-100clust_network.txt

Table Panel

shared name	name	member_count	seq_count
1	1	7362	29436

To visually select a number of nodes, hold down ‘ctrl’ on the keyboard and hold down the left mouse button. Then drag to select a number of nodes. Click any blank area in the network to deselect all nodes.

Once the user has selected one or more nodes, they can also select all nodes connected to the selection by using the ‘First Neighbours of Node’ button on the top tool bar. This can be clicked repeatedly to keep adding connected nodes to the selection.



The visual selection of nodes is useful for a fully annotated network, where the user can identify interesting nodes or regions in the network. However, when starting to annotate a network, the user does not know which nodes to target specifically. In addition, some applications require a systematic way to select nodes, such as searching for nodes that have a certain attribute or applying annotations.

The most basic way to select nodes using specific information is to select the node through the ‘Node Table’. To start, the user can select any node in the network by clicking on a row in the table (or drag to select a series of rows), then right click and choose ‘Select nodes from selected rows’. This will achieve the same effect as clicking.

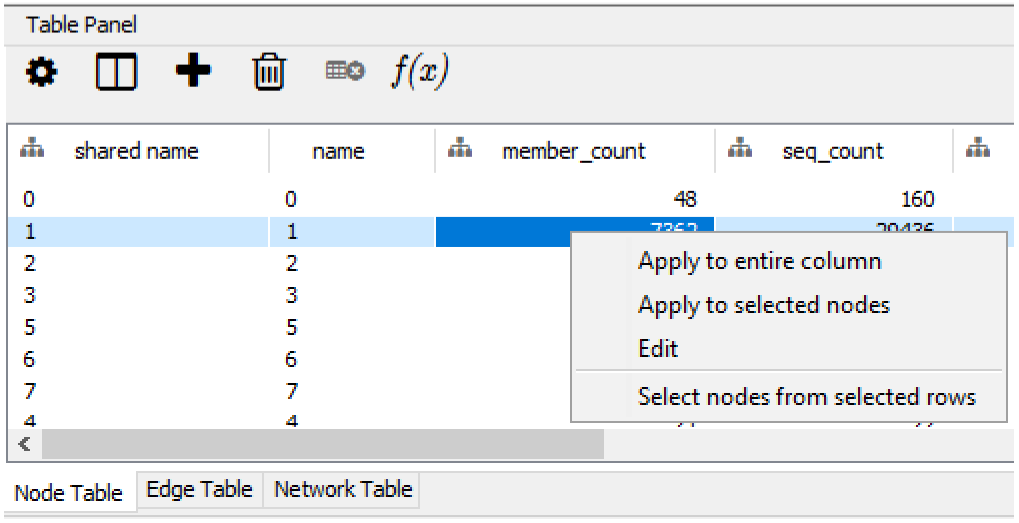


Table Panel

shared name	name	member_count	seq_count
0	0	48	160
1	1	7362	20436
2	2		
3	3		
5	5		
6	6		
7	7		
4	4		

For certain types of data, the user can take advantage of the sorting feature in the 'Node Table'. By clicking on a column, the entire table will be sorted by the values in that column. This can be an easy way to select certain data. For example, since most of our annotations are numerical, we can click any of the 'count' annotation columns to sort the table by the highest counts, then select just the nodes with a count of at least 1.

To do this, navigate to the network at thresholds of 150-200, by clicking it in the 'Network' tab of the control panel. Then, click the column in the 'Node Table' for 'sequence:classA_SBL_count', which is the annotation that shows how many known class A SBL sequences are within each metanode. Once the largest values are at the top, select all rows in the table with a value of at least 1 and select the nodes.

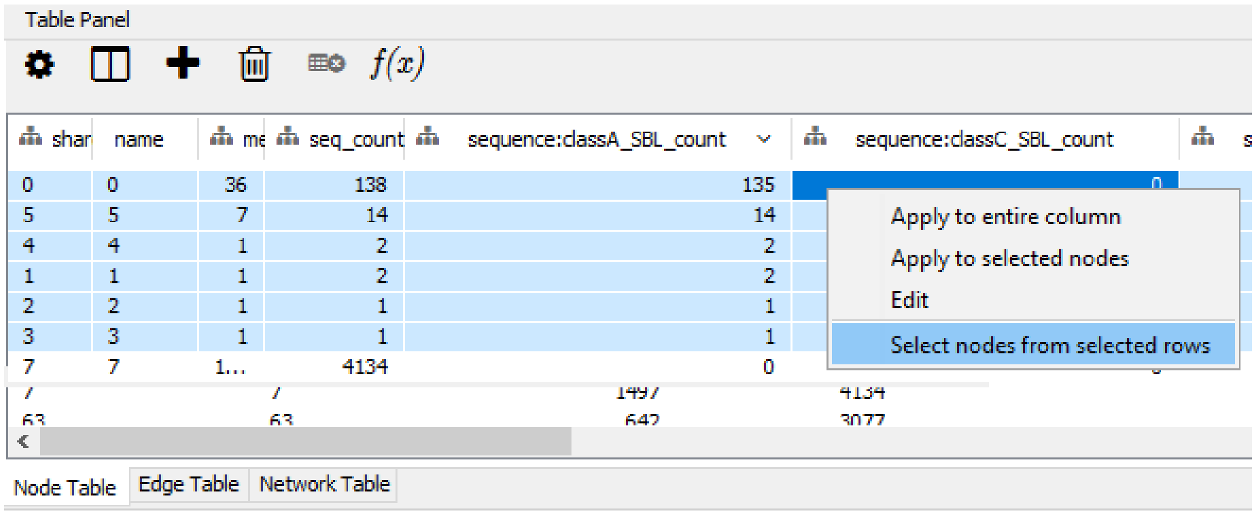
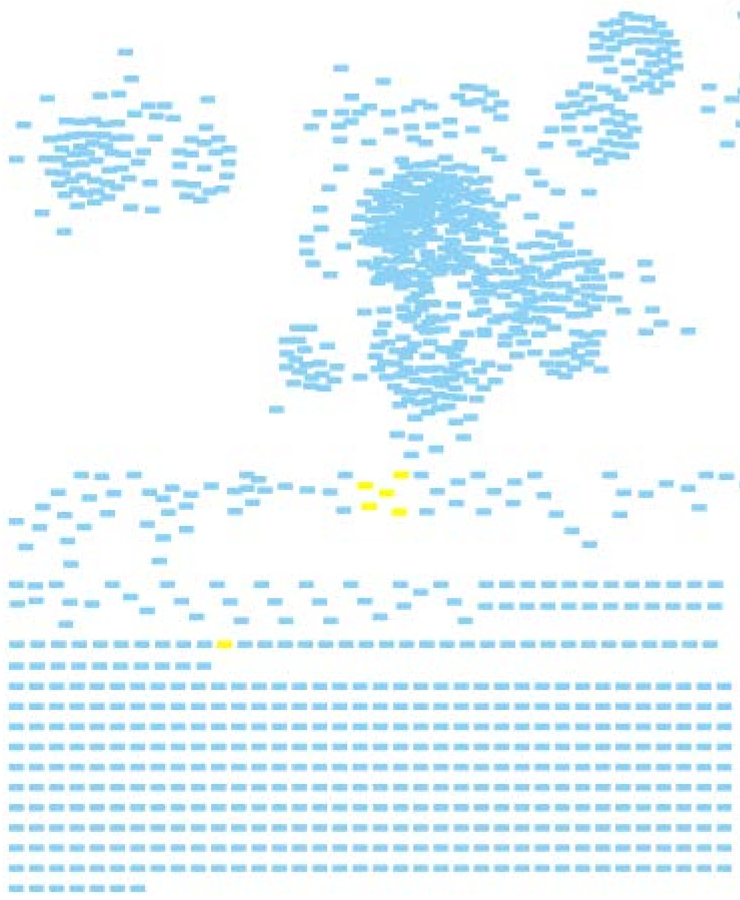


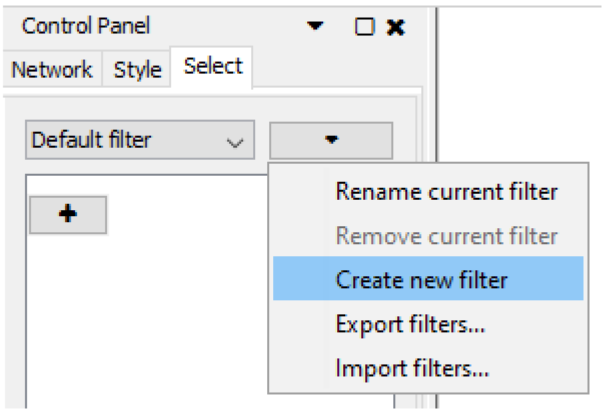
Table Panel

shar	name	me	seq_count	sequence:classA_SBL_count	sequence:classC_SBL_count
0	0	36	138	135	0
5	5	7	14	14	
4	4	1	2	2	
1	1	1	2	2	
2	2	1	1	1	
3	3	1	1	1	
7	7	1...	4134	0	

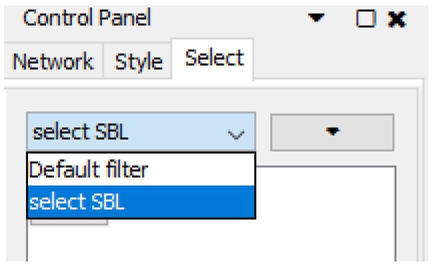
The selections can then be seen within the network.



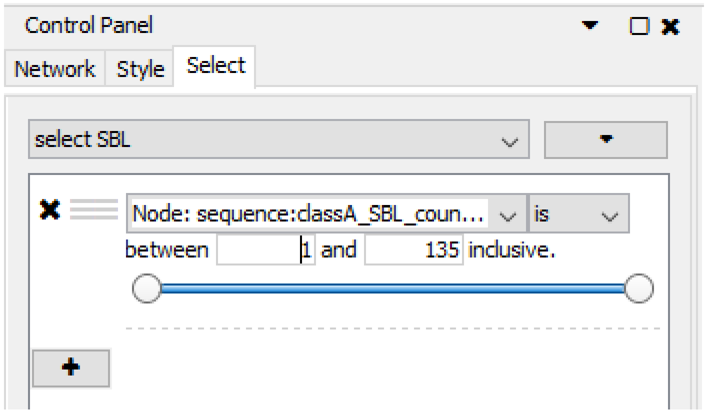
To conduct a search of nodes in a systematic fashion, we will use the ‘Select’ tab in the control panel. This panel allows the user to construct a filter for certain sequences of interest. The search filters can also be saved and reused. To start, generate a new filter by clicking on ‘Options’ (downward triangle) under the ‘Select’ tab of the control panel.



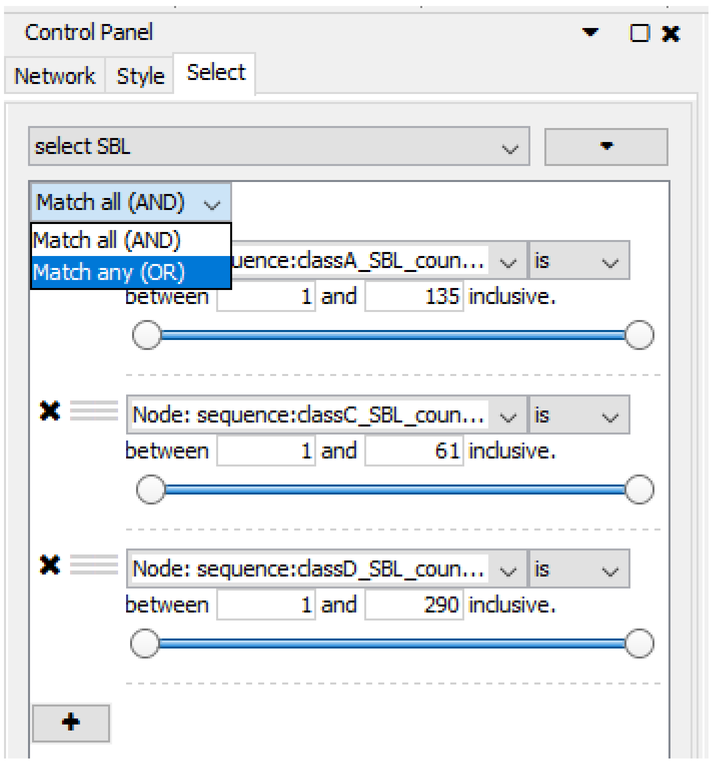
Name the filter to ‘select SBL’. Notice that the dropdown menu beside the options now shows the new filter ‘select SBL’ instead of ‘Default filter’. The user can click on the dropdown to see the list of filters that have been stored and click on the name to load that filter.



As an example, we will replicate the selection we made on ‘sequence:classA_SBL_count’. Click the plus icon to add a new filter condition, then select ‘Column Filter’. In the new dropdown box, select ‘Node: sequence:classA_SBL_count’. A search filter is now established that filters nodes based on the number in this column. Set the number in the left box (0 by default) to 1, and the six nodes we previously selected should now be selected again.



We can extend this search filter to search for all metanodes with at least one known SBL sequence, by adding additional search filters. Use the plus button to set up the same filters for the class C and class D columns, just like the class A column. Then we will set the search logic to be an ‘OR’ operation, meaning a node will be selected if any one of the filters is met.



select SBL

Match all (AND)

Match all (AND)
Match any (OR)

uence:classA_SBL_coun... is
between 1 and 135 inclusive.

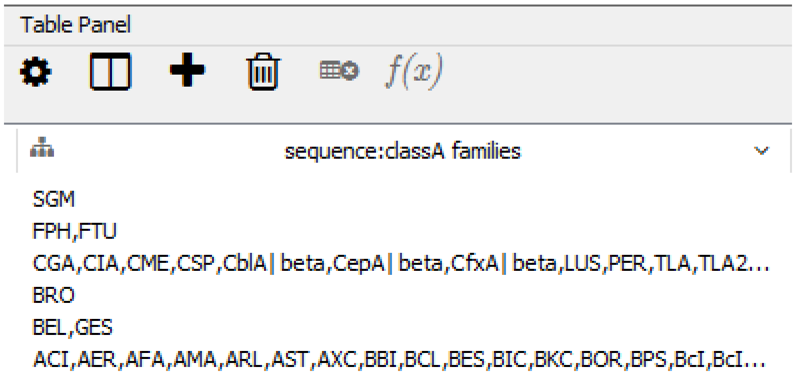
Node: sequence:classC_SBL_coun... is
between 1 and 61 inclusive.

Node: sequence:classD_SBL_coun... is
between 1 and 290 inclusive.

This new filter should select all nodes with a known SBL sequence in it.

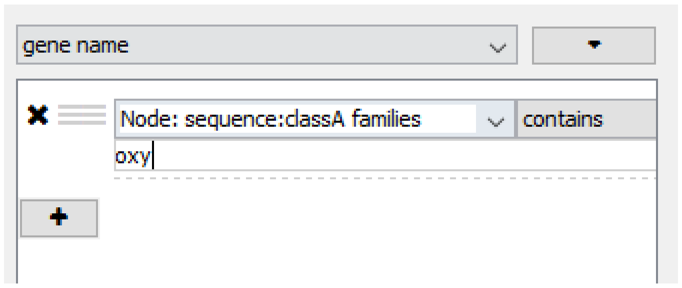
It is important to mention that searches and selections can be conducted in multiple ways. Recall during the MetaSSN configuration, we established a ‘count’ annotation that combined all known sequences from the class A/C/D SBLs. Using this column, we can also make an equivalent selection to the one we just constructed by sorting the column. An important consideration is to think about ways that searching and selection could be made easier if already incorporated into the ‘Node Table’, and trying to build annotations this way could be easier than applying a filter. However, the ‘Select’ tab in the control panel can be immensely useful for annotations that are not easy to aggregate in such a way, such as annotations that contain specific gene names.

To showcase a search filter that looks for a specific gene name, we will make a new filter called ‘gene name’ and add a column filter for the ‘sequence:class A families’ column. As can be seen, the column contains the names of class A SBL families that are present within a metanode. However, the tabular format is insufficient to display all names at once when there are many entries, even for short abbreviate names that we have supplied.



SGM
FPH,FTU
CGA,CIA,CME,CSP,CblA| beta,CepA| beta,CfxA| beta,LUS,PER,TLA,TLA2...
BRO
BEL,GES
ACI,AER,AFA,AMA,ARL,AST,AXC,BBI,BCL,BES,BIC,BKC,BOR,BPS,BcI,BcI...

We can make a simple filter to look for the OXY-type class A genes as shown below.



gene name

Node: sequence:classA families contains

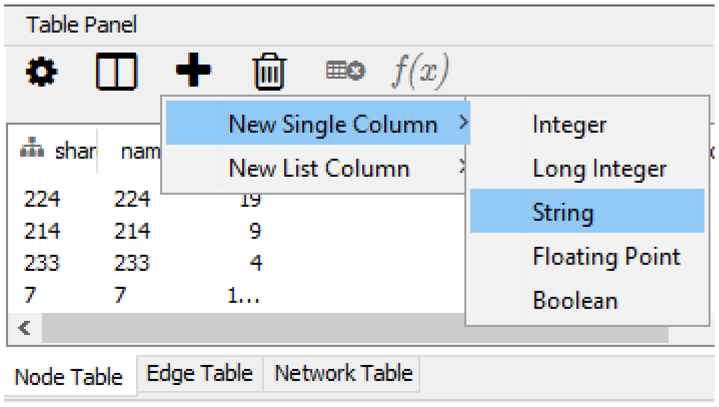
oxy

6.1.3 Adding annotations to selected nodes

Though we have supplied a variety of annotations to the network that facilitates searching and navigation, some additional processing is required to make visualizations. This is because most annotation options rely on information from a single column, while most of our annotations are kept in separate columns for the purpose of being able to distinguish different properties within a metanode. For example, the number of known class A/C/D SBLs within a node can be used to construct a total number of known SBLs in a node, but knowing the total number of known SBLs within a metanode provides no information on which specific families are present.

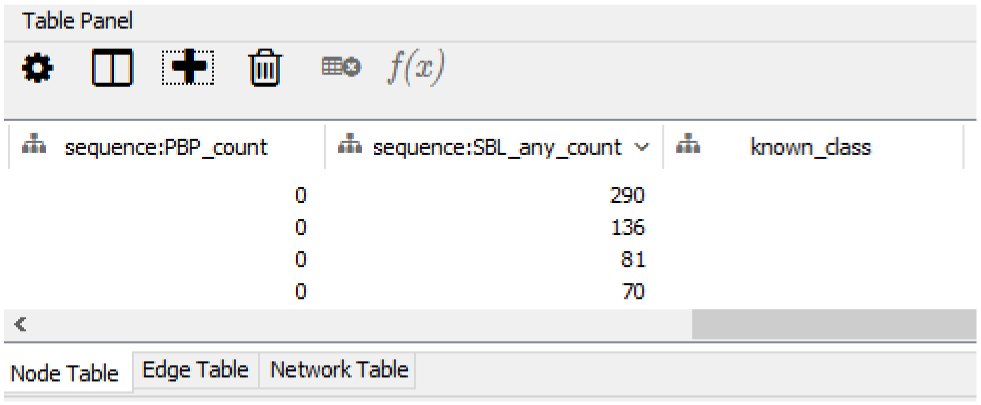
Recall that the SBL sequences in the network are those with the ability to degrade beta-lactam antibiotics while the PBP sequences are those that are the natural inhibitory targets of beta-lactams. If we wished to generate visual styles to identify the SBL and PBP using different values within a certain class of visual characteristics (shape, color, etc), the information that identifies a metanode containing SBLs or PBPs needs to be combined into a single column.

We can start by creating a new column. Ensure that the network with thresholds of 150-200 is selected. Go to the table panel and make sure the 'Node Table' is selected, then click the plus sign in the table panel to create a new column. Select 'New Single Column' then 'String'.



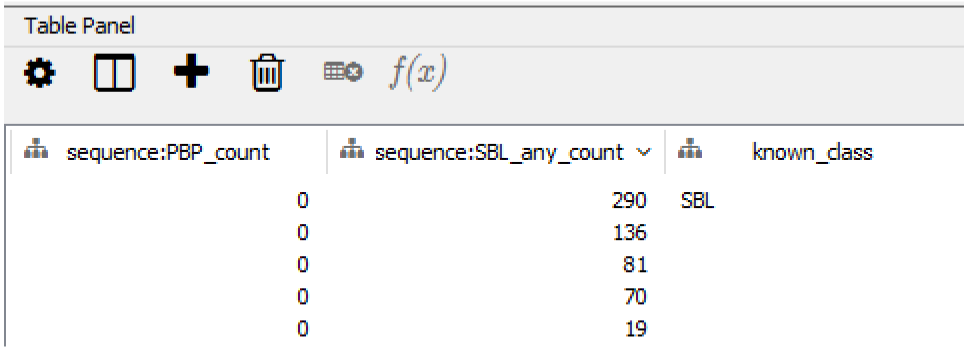
shar	nam	
224	224	19
214	214	9
233	233	4
7	7	1...

Name the column 'known_class'. This column will be generated at the right end of the table. Move this column to a more convenient location by clicking and dragging on the column header. Move it next to the 'sequence:SBL_any_count' column, which is close to where we will perform selections.



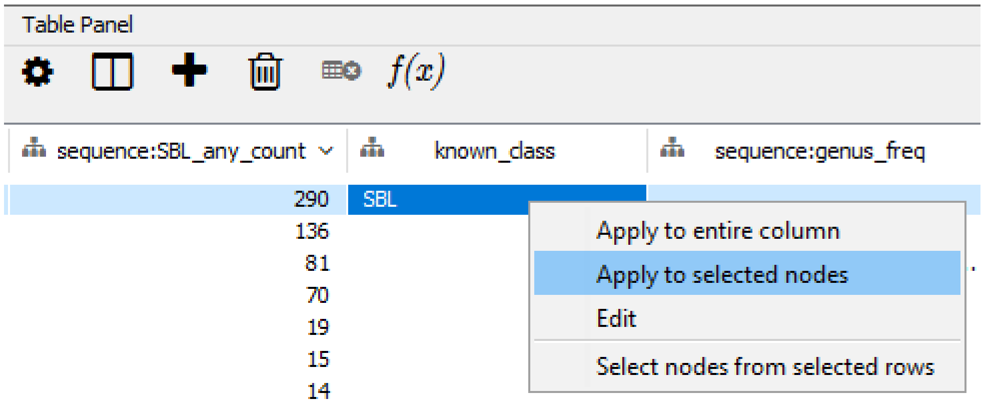
sequence:PBP_count	sequence:SBL_any_count	known_class
0	290	
0	136	
0	81	
0	70	

Since we already have a column for both the count of SBL and PBP sequences within a single metanode, we can easily select nodes with at least one sequence of each category by sorting the column in the table. Sort the 'sequence:SBL_any_count' column and select all rows with at least 1 known SBL, so that only the selected rows are present in the table. Then double click any selected cell in the 'known_class' column to edit the value to 'SBL'.



sequence:PBP_count	sequence:SBL_any_count	known_class
0	290	SBL
0	136	
0	81	
0	70	
0	19	

After, right click the cell you filled and select 'Apply to selected nodes'. All the nodes with at least 1 known SBL sequence will now be labelled 'SBL' in this new column.



sequence:SBL_any_count	known_class	sequence:genus_freq
290	SBL	
136		
81		
70		
19		
15		
14		

Deselect the SBL nodes. Repeat the selection for the 'sequence:PBP_count' and label the selected nodes 'PBP'. This column can later be used to visually label the nodes.

As usual, there are multiple ways to annotate and select nodes in a network. One way to achieve the same information using MetaSSN instead of Cytoscape is to use the 'label' annotation, and simply supply a list of all SBLs whose labels are all 'SBL' and also supply a list of all PBPs whose labels are all 'PBP'.

To showcase another type of operation that we can do, we will convert the numbers in the 'seq_count' column (number of starting sequences in each metanode) to a Log10 scale. The count of sequences within a node is a common feature used to style the network, so that the user can gauge how many sequences are within a region of a network. However, with an attribute like 'seq_count' there can often be a large discrepancy between the number of sequences between metanodes, making the visualized node sizes essentially indistinct as nodes with smaller numbers are harder to see. To adjust for this, we can convert the numbers to log scale to make order of magnitude differences appear as linear.

Again, create a new column, but this time select New Single Column > Floating Point. Name the column 'log10_seq_count'. Click any cell in the column and type the following formula:

```
=LOG($seq_count,10)
```

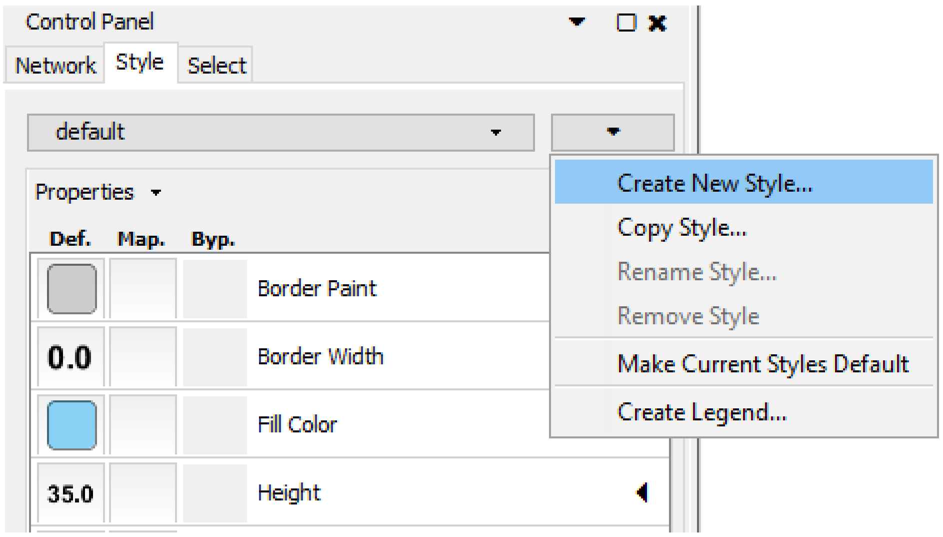
Right click and select 'Apply to entire column'. As can be seen, Cytoscape implements some basic operations into the table panel using a fashion similar to Excel. For a list of such function, check the Cytoscape manual.

One point to note is that manual addition of annotations have to be done separately for each network at each set of thresholds. Thus if you are expecting to work with 10 or more networks, these annotations should be incorporated as an automated step if possible. Otherwise, one can filter for networks of interest based on the data in each network before trying to visualize them (see Section XX).

6.1.4 Applying visualizations based on annotations

We will apply some basic visualizations to the network. Ensure you are using the 150-200 threshold network as in the previous steps.

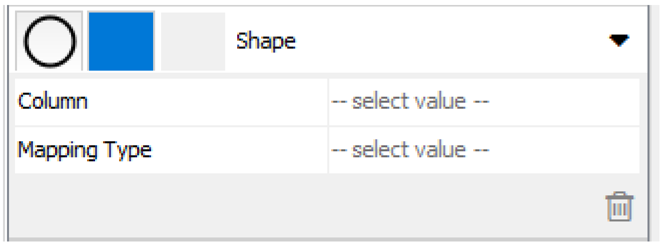
Navigate to the 'Style' tab of the control panel. The 'Style' tab functions very similarly to the 'Select' tab, in that you can define settings that can be saved for repeated use. Again, click the downward arrow near the top and select ‘Create New Style’ to create a new style template. Name this template ‘shape_known’.



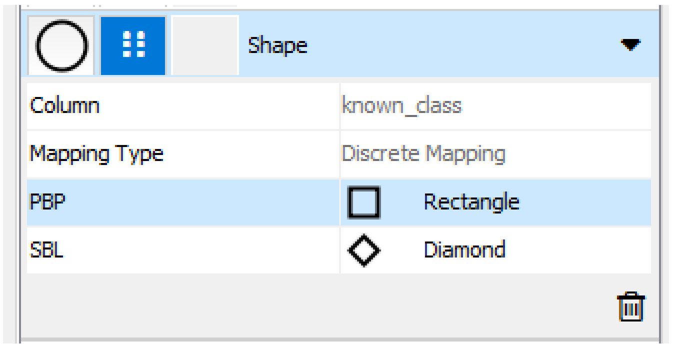
The newly created style will be applied automatically. As with before, the styles can be easily changed using the dropdown at the top. The default when creating a new style is to set the default color to red, default shape to circles and the width and height of each node is ‘locked’ (when one is changed both are changed). As can be seen at the top, each styling has 3 settings, a default (Def.), a mapping (Map.) and bypass (Byp.).

To illustrate how to change the default styling (default styling is applied if the node does not receive any other specific style), click on the red box in the ‘Fill Color’ section in the ‘Style’ menu. Note the red box is under the ‘Def.’ column of stylings. When the color selection menu appears, select a shade of grey. Grey is a good neutral color for the default, because it is easy to distinguish from any colors that are specifically applied.

Next to illustrate how to set a mapping based on an annotation column, go to the ‘Shape’ section and click on the middle box under the ‘Map.’ Column. You will see a new dropdown appear.

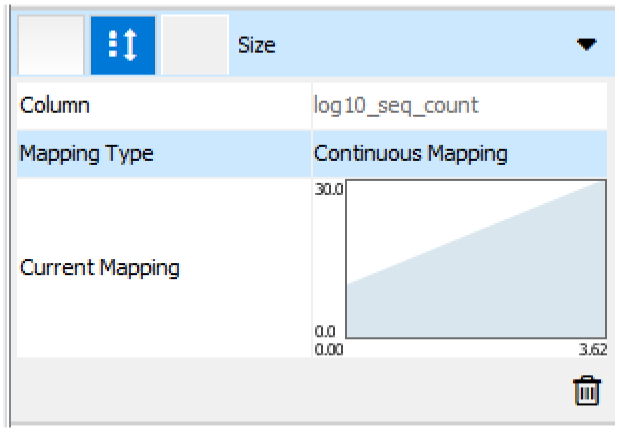


The mapping menu is shared by all the stylings and contains the ‘Column’ option, which sets the column in the ‘Node Table’ to use for annotation data, and the ‘Mapping Type’. Click on the column option and select the ‘known_class’ column that we have manually added. For the ‘Mapping Type’ select ‘Discrete mapping’. This will cause each non-empty value in the ‘known_class’ column to become a category for mapping stylings to. In this case, set the ‘PBP’ to ‘Rectangle’ and the ‘SBL’ to ‘Diamond’, as shown below. Notice that all the relevant nodes have changed shape in the network, where nodes with at least one known PBP are rectangular and nodes with at least one known SBL are diamond shaped.



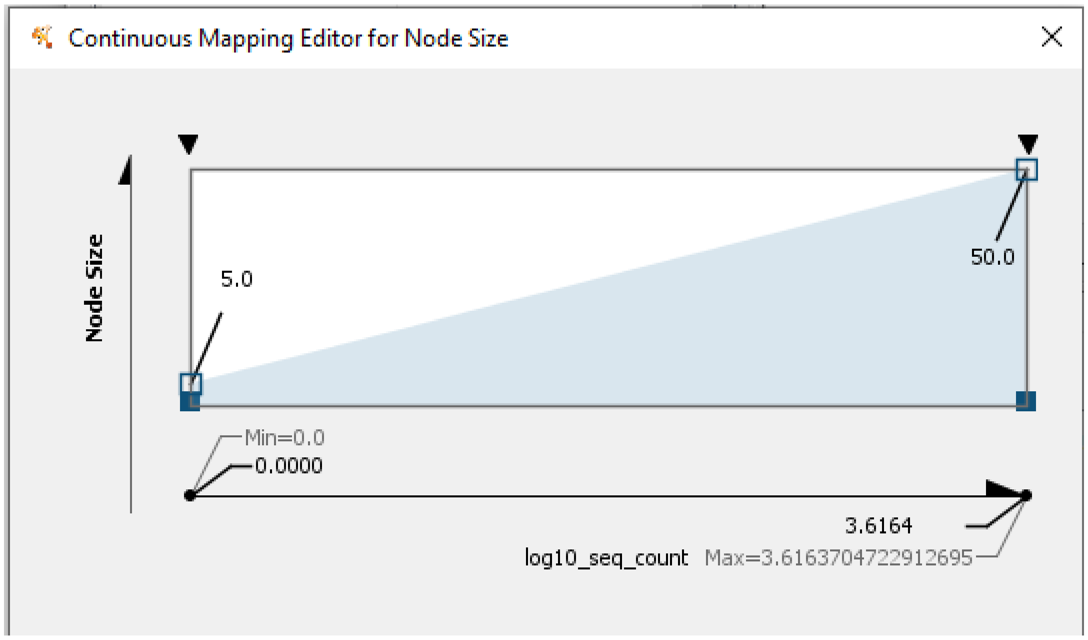
Column	known_class
Mapping Type	Discrete Mapping
PBP	Rectangle
SBL	Diamond

Next, we will adjust the size of each node to be based on the number of sequences in a metanode. Select the square in the 'Map.' column of the 'Size' section of stylings. Set the 'Column' setting to the 'log10_seq_count' column, and select 'Continuous mapping' as the 'Mapping Type'. All node sizes will change immediately.

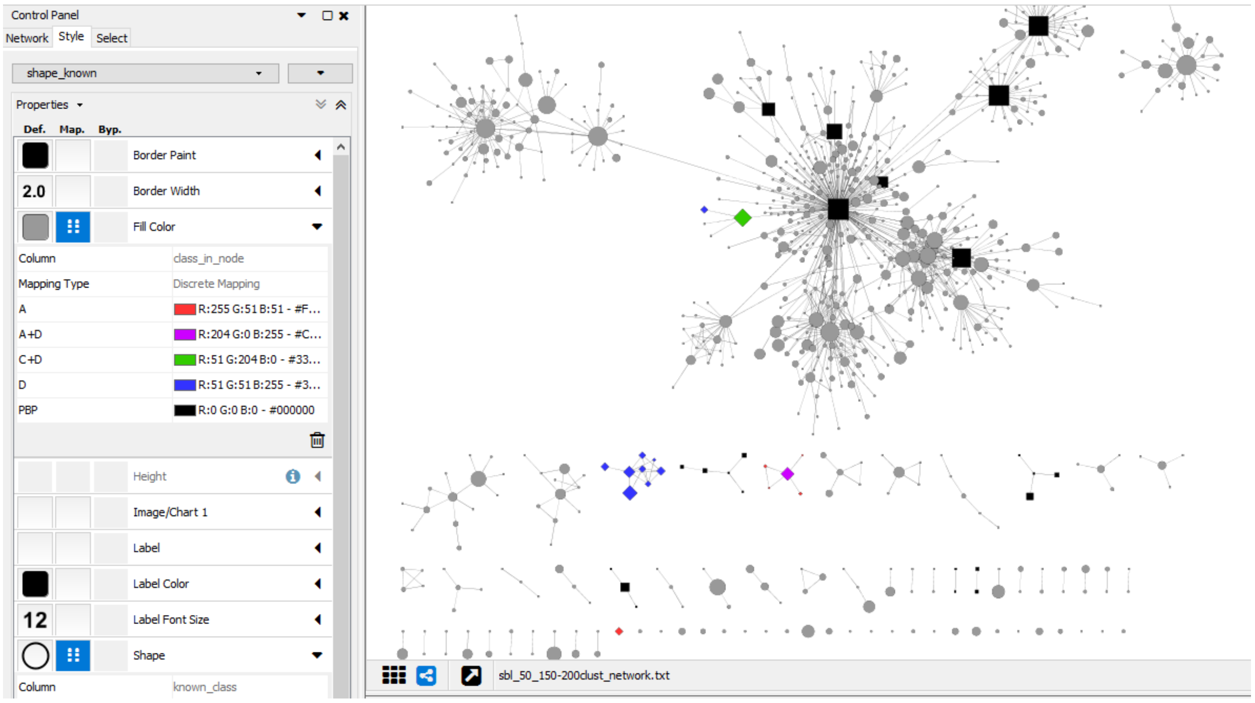


Column	log10_seq_count
Mapping Type	Continuous Mapping
Current Mapping	

By default, the minimum value will be set to a size of 10 and the maximum to a value of 30. This can be quickly visualized in the 'Current Mapping' section. Note that the data in our column has data spanning from a minimum of 0 (Log10 of 1) to 3.62 (Log10 of ~1,000-10,000). Double-click on the graph to bring up the mapping menu. The user can adjust the sizes by dragging the 'handles' or 'pivots' on the graph (hollow boxes with numbers pointing to them). To set an exact value, double-click on the 'handle' for the maximum value and set it to 100 (leave the setting for the minimum at 10). Setting a wider range will help make nodes with different sequence counts more distinct.



However, everything is still grey and nodes containing known sequences are still indistinct. I will leave it as an exercise for the user to apply a color scheme that colors each of the class A/C/D with a unique color, and also the PBPs with its own color. This will require the user to generate a new column with labels for coloring. Take note that there is a node with a mixture of classes C and D and another with a mixture of A and D. An example is shown below.



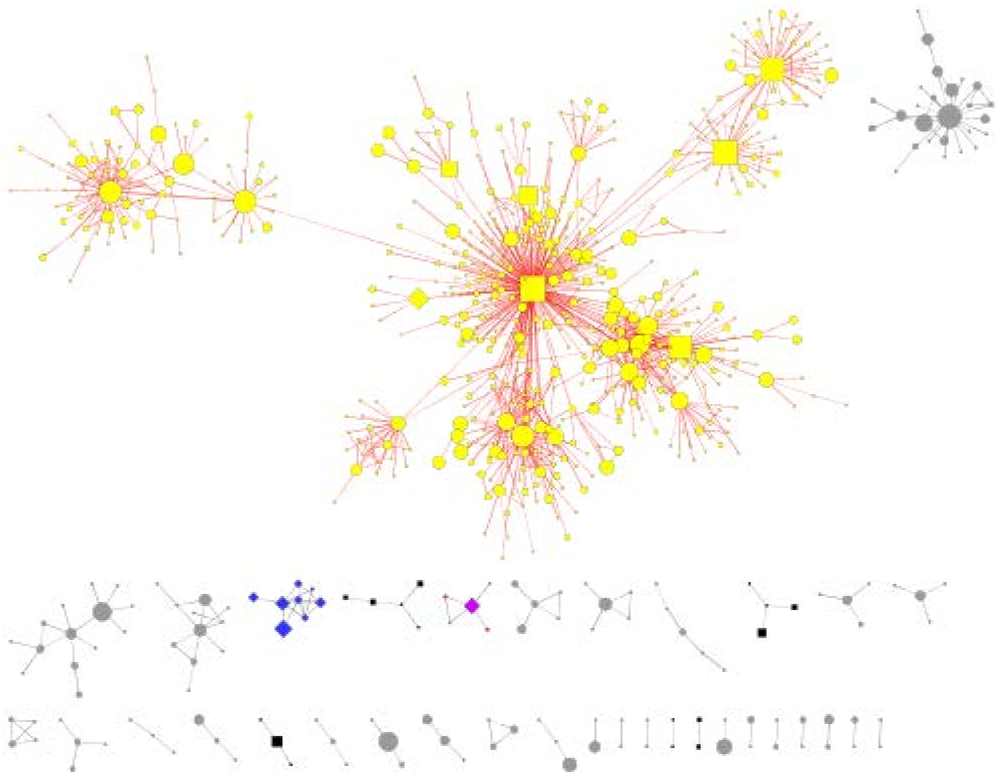
⬛	Border Paint	
2.0	Border Width	
▨	Fill Color	
Column	class_in_node	
Mapping Type	Discrete Mapping	
A	🟥 R:255 G:51 B:51 - #F...	
A+D	🟪 R:204 G:0 B:255 - #C...	
C+D	🟩 R:51 G:204 B:0 - #33...	
D	🟦 R:51 G:51 B:255 - #3...	
PBP	⬛ R:0 G:0 B:0 - #000000	
	Height	
	Image/Chart 1	
	Label	
⬛	Label Color	
12	Label Font Size	
○	Shape	
Column	known_class	

6.2 Network manipulation

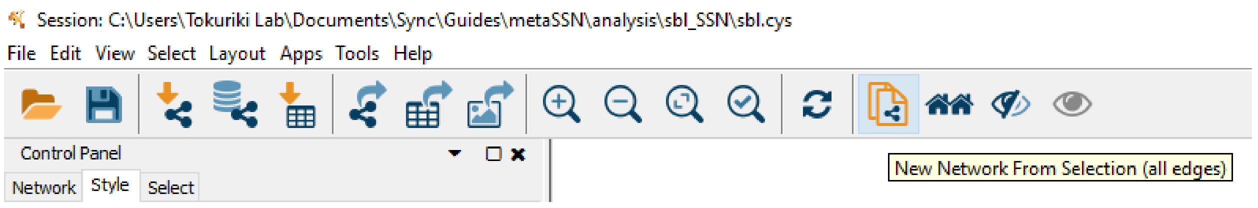
6.2.1 Selecting subnetworks in CytoScape

Often times, only a portion of the network is of interest to the user. The methodology of the SSN tries to explore sequence relationships over a broad set of sequences, which can be very helpful if there is no prior information on the sequence space. However, once the user has identified a region of interest (a certain set of sequences related to the user's target, a certain region with uncharacterized sequences, etc.), they can start to hone in on just the part of the network that they are interested in. Extracting a 'subnetwork' helps the user focus their analysis and is vital for making figures that show the most important parts of the network.

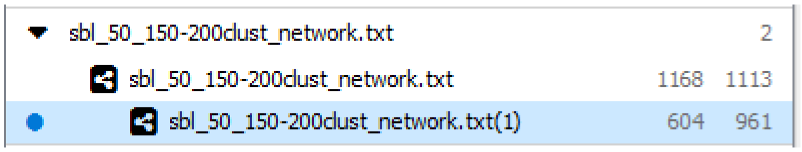
CytoScape has some of these capabilities built in. The user can simply select a number of nodes, either visually in the 'Network View' or by searching the 'Node Table', and collect just those nodes into a new network. To illustrate, we can select the largest interconnected set of nodes in the 150-200 threshold network.



Then simply click the 'New Network from Selection' button in the top tool bar, and a new network will be made containing just the selected nodes.


Session: C:\Users\Tokuriki Lab\Documents\Sync\Guides\metaSSN\analysis\sbl_SSN\sbl.cys
File Edit View Select Layout Apps Tools Help
Control Panel
Network Style Select
New Network From Selection (all edges)

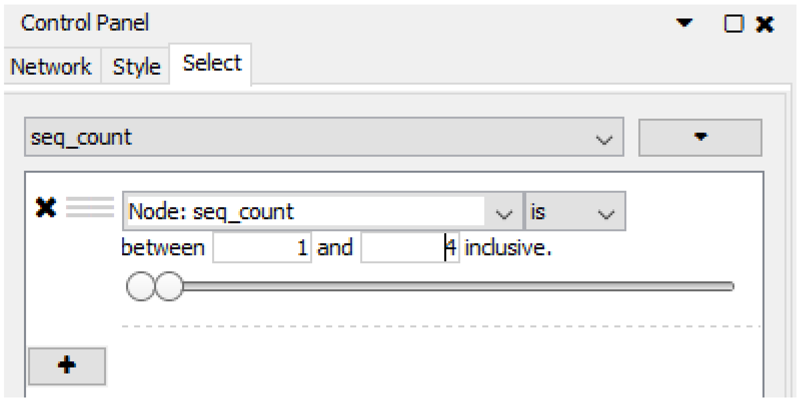
The new network will be listed in the 'Network' tab of the control panel, as a sub category of the previous network.



sbl_50_150-200clust_network.txt		2
sbl_50_150-200clust_network.txt	1168	1113
sbl_50_150-200clust_network.txt(1)	604	961

6.2.2 Hiding and deleting nodes in CytoScape

Sometimes it is useful to hide or remove nodes from a network to remove clutter. To illustrate this, select the original network at a 150-200 thresholds. We will make a copy of this network, and remove all metanodes with less than 5 starting sequences in it. Select all the nodes in the network using 'ctrl+A' and create a subnetwork using the 'New Network from Selection' button in the top toolbar. In this new network, use the 'Select' tab in the control to select all sequences with less than 5 sequences. As shown below.



seq_count

Node: seq_count is between 1 and 4 inclusive.

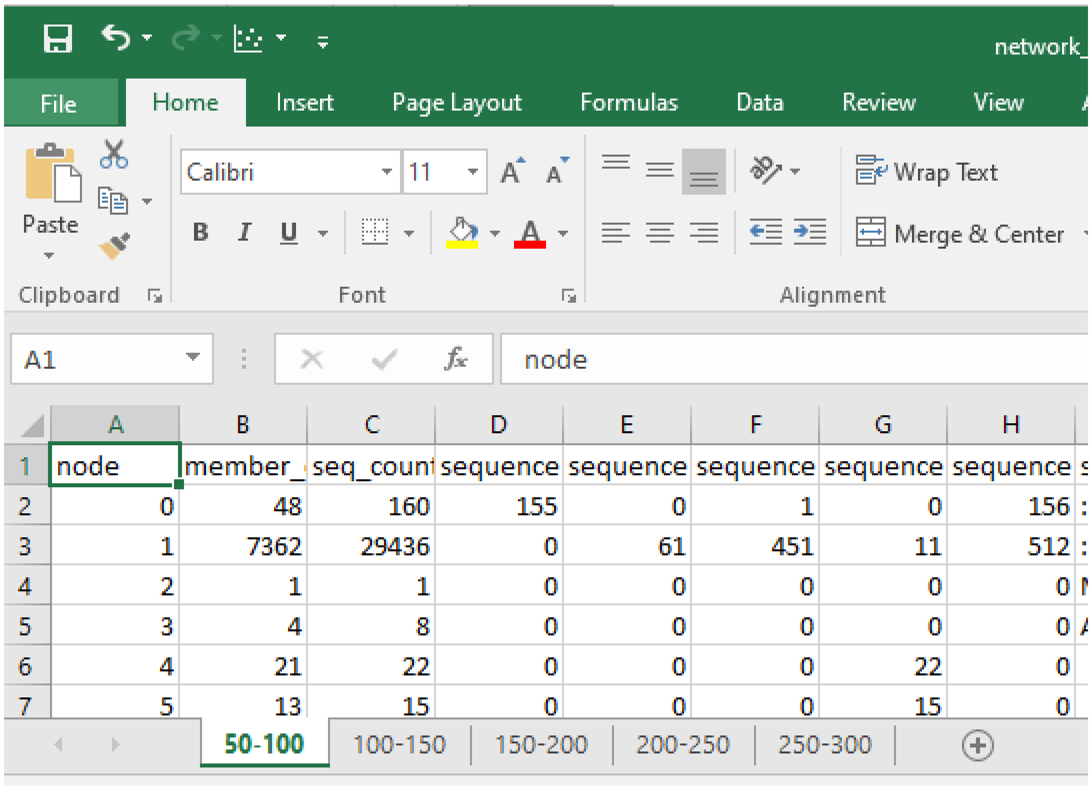
If the selection is not made automatically, hit the 'Apply' button at the bottom. From here simply press the 'Delete' key on the keyboard or navigate to 'Edit' in the top menu and select 'Delete Selected Nodes and Edges'. To reorganize the network, navigate to 'Layout' in the top menu and select 'Perfuse force directed layout' > (none).

To hide the selection instead of deleting it, navigate to 'Select' in the top menu and select 'Hide Selected Nodes and Edges'.

6.2.3 Examining networks using node information

If the user has no prior information on which set of network thresholds would be most informative, they can opt to do a pre examination of the network by simply looking at the network table. This can be useful if the networks are large and slow to handle in Cytoscape, or if the user simply has a large number of networks (such as doing a high resolution sweep of the network at bitscore intervals of 10). The simplest way is to copy and paste, or open, the '_node_summary.txt' files in excel. From here, the same operations that the user would conduct on the 'Node Table' in cytoscape can also be used to examine the networks. For the tutorial, we will open each of the node summary files in excel, and sort each table by the number of known class A SBLs in each metanode.

Start by loading each network's node summary into a different work sheet in an excel work book. As shown below.



A1 node

	A	B	C	D	E	F	G	H
1	node	member_	seq_count	sequence	sequence	sequence	sequence	sequence
2	0	48	160	155	0	1	0	156
3	1	7362	29436	0	61	451	11	512
4	2	1	1	0	0	0	0	0
5	3	4	8	0	0	0	0	0
6	4	21	22	0	0	0	22	0
7	5	13	15	0	0	0	15	0

50-100 | 100-150 | 150-200 | 200-250 | 250-300

In each sheet, select the entire table using 'ctrl+A' on the keyboard, then navigate to 'Data' in the top menu of Excel. Then select 'Sort' and select 'sequence:classA_SBL_count' as the target for 'Column', leave 'Sort On' as 'Values', and set 'Order' to 'Largest to smallest'. When sorting, always select the entire table, or else a single column could be sorted independently of the table and cause all the data to become misaligned.

This allows a cursory overview of the network in terms of where the class A members are distributied. For example, at 50-100 thresholds, the class As are already separated into its own node away from all other known classes of SBLs. At 100-150, the class A sequences split into a major node with 141 sequences (ACI-type, AER-type, etc.) and a smaller node containing 14 sequences (CGA-type,CIA-type, etc.). The next major separation within the class A occurs at 200-250, where another node (containing CM1, LEN-type, etc.) separates from the main class A node.

As can be seen, this can be a convenient method to prescreen the networks if the user has a particular purpose in mind, such as looking for a specific separation point between certain sequences. The user can then go back to MetaSSN and generate more networks between the thresholds of interest to further interrogate the separation in more detail.

An alternative to screen a network without loading them into Excel is through the command line. A Unix command line tool called 'awk' can be easily used to read through and filter tables. The following command will take a node summary table and print out just the 'node','sequence:classA_SBL_count' and 'sequence:classA_families' columns.

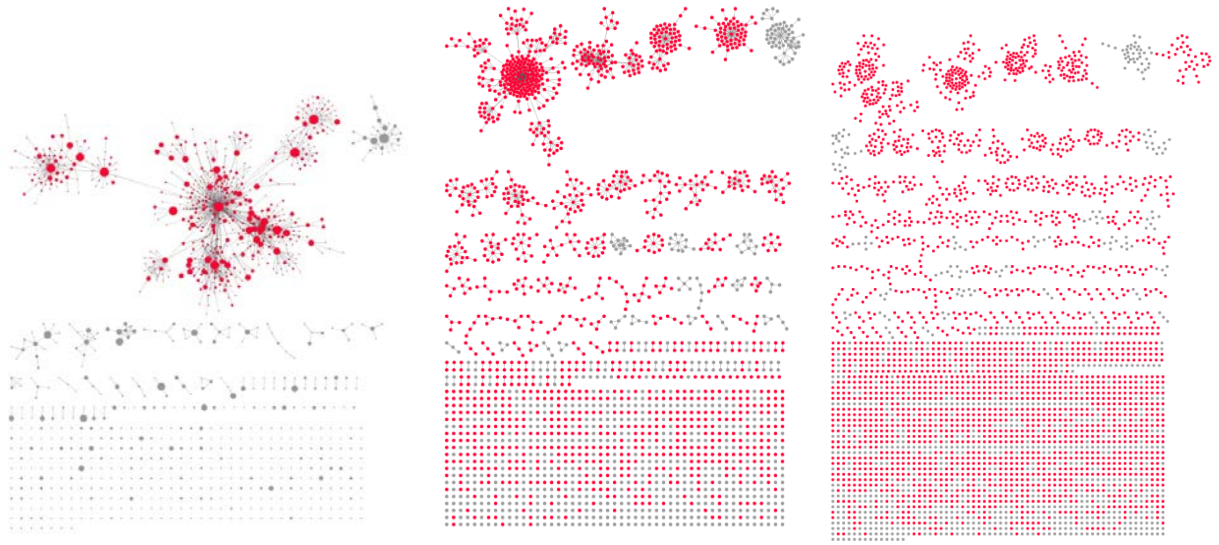
```
awk -F $'\t' '($4>0){print $1"\t"$4"\t"$12}'
sbl_50_100-150clust_node_summary.txt
```

6.2.4 Extracting subnetworks at different cut-offs by node membership

One common use of SSN analysis is to split a superfamily into 'subgroups' which contains a subset of similar sequences. The sequences could contain sequences of known function, allowing the user to guess the closest functional link to the sequences in the subgroup. Sometimes, one would like to examine the further propagation of the networks at a higher thresholds within a certain subgroup, but these will start to separate into different nodes and may not be connected by edges past a certain threshold. It makes it difficult to select the members of a subgroup visually.

One way to make a certain subgroup easy to search is through the use of an extra annotation that keeps track of the nodes in a more separated network in terms of their membership in a less separated network. In this case, the less separated network would be the level at which a certain subgroup is defined. To illustrate, we can look at the largest node in the network at 100-150 thresholds (node index of 3), and see that it still contains the class C sequences and a small set of class D and PBP sequences. To see how this main node further separates, we can track which nodes at higher thresholds belong in this main node.

To do this, we can make use of the 'membership_lower_node' option that we configured in **section 5.5.2**. This annotation tells us which of the nodes at a lower clustering threshold use to contain the nodes in the current network at a higher clustering threshold. We can create a new visual style and color just the members of the main node, node 3, at the 100-150 threshold. In the images below, you can easily see all nodes that descend from node 3 at the 100-150 thresholds colored in red (thresholds are 150-200, 200-250, and 250-300 from left to right).



Another way to be do this is to simply decide the clustering threshold for which the subgroup is defined and then generate subsequent networks using that threshold as the edge threshold. Recall that the clustering threshold defines whether sequences are collected into the same metanode and the edge threshold determines if different metanodes are connected based on the BLAST bitscore of their members. When the edge bitscore is equal to the clustering bitscore of a lower network, the nodes that descend and separate from each node in the lower network will continue to be connected at the higher networks. This can be examined through the leftmost image above, where the descendents of node 3 at 100-150 thresholds all form an interconnected subnetwork. The user can further increase the clustering threshold while maintaining the same edge threshold to see further network separation while still keeping all the nodes within the same subgroup connected.

Finally, there is a helper script that comes with MetaSSN designed to extract all nodes that contain any members from a target node in a specific network. To see how to use the 'fetch_subnetwork.py' script, see **section 5.1** in the MetaSSN manual.

6.3 Extraction of sequences from the network

The chief advantage of an SSN is the ability to visually display the connections between sequences within a superfamily level. The SSN is a starting point for exploration of sequence space, and other methods of analysis would be needed to further examine the sequences within the network. As such, once the user has identified certain regions or clusters in the network, they would need to extract the sequence identifiers or the sequences themselves for further analysis.

If the user simply wishes to identify which sequences are within a specific set of nodes, they can extract all 'members' or all 'sequences' from the '_membership.txt' and '_nodes_all_seqs.txt', respectively.

To demonstrate this, we will make a simple text file with a list of nodes from the 100-150 threshold network. We will extract an arbitrary set of nodes, taking node 0 (all the class A sequences), 34 and 160 (all the class D sequences). In practice, the user should perform separate extractions if they want to keep the sequences separate, i.e. to distinguish different families. Name the file 'target_nodes.txt' and save it inside the 'analysis/sbl_SSN' folder (same location as the network files). The file should look like the following, with one ID per line.

```
0
34
160
```

The following Python script takes a file containing a list of node IDs (one per line) and extracts all members within the specified nodes. Run it inside the 'analysis/sbl_SSN' folder.

```
extract_one_to_many_mapping.py   target_nodes.txt
sbl_50_100-150clust_node_all_seqs.txt    extracted_seq_IDs.txt   -C
```

This collects all sequences in the target nodes in 'target_nodes.txt' based on the node-sequence mapping inside the 'sbl_50_100-150clust_node_all_seqs.txt'. The extracted sequences are placed inside the 'extracted_seq_IDs.txt'.

If the sequences in the network were already labelled by their original identifiers, the list can be used to further identify and analyse sequences. However, if the sequences have been converted to numerical indices, a further conversion using the following Python code can be used to retrieve the full sequence headers.

```
convert_mapping.py  extracted_seq_IDs.txt
'../../data for BLAST/sbl_all_map'  extracted_seq_headers.txt  -C
```

The converted headers can be found inside the output file ‘extracted_seq_headers.txt’.

To extract the actual sequence data, a similar procedure can be used. In the case of this tutorial, all the sequences have been merged into a single file whilst simultaneously converting the sequence headers into numerical indices. The network contains sequences information in terms of the numerical indices and it would be most direct to extract the sequences using the indices. The file ‘extracted_seq_IDs.txt’ can be used as input for the ‘extract_fasta_by_header.py’ script, as shown below.

```
extract_fasta_by_header.py  extracted_seq_IDs.txt
'../../data for BLAST/sbl_all.fasta'   extracted_seq_IDs.fasta
```

The extracted sequences are stored in ‘extracted_seq_IDs.fasta’, with the headers still in numerical indices. To convert them, use the ‘rename_fasta_headers.py’ script.

```
rename_fasta_headers.py   extracted_seq_IDs.fasta
'../../data for BLAST/sbl_all_map' extracted_seq_headers.fasta
```

Sequences with converted headers can now be found inside ‘extracted_seq_headers.fasta’. Of course, the user can use the ‘extract_fasta_by_header.py’ directly if the headers already correspond to the original sequence headers, without need for converting the headers afterwards.