Meta-SSN Manual

Contents

1. Overview

1.1 Sequence similarity networks and drawbacks of current approaches

A sequence similarity network (SSN) is a tool often used for visualization of sequence relationships within a protein superfamily. The method is applied by conducting an all-by-all BLAST on a dataset of protein sequences, and making a network where protein sequences (represented as nodes) are connected (by edges) if they share similarity (BLAST bit score) above a given threshold. A force directed visualization algorithm is then applied, causing regions of highly connected sequences to visually cluster. In this way, an SSN is an intuitive way to explore protein superfamilies, where highly similar sequences will appear as densely interconnected clusters.

Compared to phylogenetic methods which can also be used to explore interrelationships of sequences within a superfamily, an SSN is computationally cheap to generate and supports a much larger dataset, but at the cost of having no inference on the model of evolution between sequences. Even so, the rapid expansion of known protein sequences has made SSNs hard to manage. The main issue lies in the number of edges that a network could have, since the number of results generated by an all-by-all BLAST of 'N' sequences is between N^1 to N^2 and grows exponentially. Thus even a modest number of sequences can generate a massive number of edges (5000^2=6.25million), to the point where

visualization and manipulation on a normal computer is slow and or impossible. The majority of these edges are also highly redundant, creating extremely dense interconnection between very similar sequences (which provide non-informative relationships, since all it highlights is that the sequences are highly similar and could be generated with just a BLAST search) while connections between less similar sequences are sacrificed to provide space for the high similarity edges (much more detrimental, since connection between subfamilies is the most interesting information provided by an SSN). The simplest way to try and avoid this problem is to reduce the number of sequences and collapse highly similar sequences into representative sequences using tools like CD-hit. However, even clustered networks can easily surpass sequence counts of 5000-10000 and in the continuing growth of sequence data this is a delaying tactic at best. Another way to overcome this issue is to cluster highly similar sequences into the same node, thus eliminating all redundant edges from the visualization while still retaining the perceived "clustering" of these sequences {should ref Brumer lab software, but apparently not published yet}. Since this is the generation of a "summarized" network from the base network, it is called a "meta-network", and similarly the new "meta-nodes" in the network are a combination of nodes in the base network.

A new problem that arises is that the meta-nodes are now a collection of nodes rather than a single sequence, each of which could also be a representative to a further set of sequences. To be able to annotate these meta-nodes in the network requires keeping track of all members and assembling the annotations into each node through some summary method. The existing example is the EFI webtool for SSN generation, which extracts the UniProt attributes for sequences included in the network, and also compiles information from member sequences if the nodes are representative sequences. Unfortunately, the EFI tool has a few drawbacks:

1. It only applies this approach when data is given through predefined sequence definitions such as pfam family IDs. When custom collections of data are given, the annotations are not included (may have changed, don't know).
2. EFI can only add pre-existing attributes from UniProt. The EFI tool cannot be used to label the network with custom data (biochemical activity, substrate preference, codon or residue at specific position), which is often generated in the process of analyzing the superfamily.
3. The attributes are collected in the most comprehensive way possible, such that each attribute is simply a concatenation of all values observed. This is hard to actually read when there are more than 5 different sequences being labelled in a node, and repeated information is not summarized for easy interpretation.

1.2 A combined tool for meta-network generation and annotation

To address some of the issues noted above, a Python based tool (Meta-SSN) is made to streamline the full process in generating and annotating SSNs. The tool is designed to allow the user access to all steps in the SSN curation process through a simple set of configurations, removing the need for repeatedly bridging the steps using separate scripts or software (not even sure anything actually does this stuff). Meta-SSN specifically handles three major tasks:

1. The generation of meta-networks, allowing for dense networks to be condensed for easy viewing.

2. Declustering meta-nodes into representative members and all sequences, so that each node can be viewed as just a group of representatives with reduced redundance or as the full sequence set to reflect all currently available sequence data.
3. Provide 5 main methods to attach annotations to each meta-node, which summarize the attributes and make them easier to interpret.

1.3 Workflow

The intended workflow for Meta-SSN can be visualized in **Fig. 1** as the steps indicated by curved arrows. All steps fit within the conventional method for SSN generation, and each is optional, allowing relatively easy integration into existing pipelines. Before Meta-SSN is applied the user will start by:

1. Collecting the data of interest from sequence databases such as UniProt.
    a. Optional: Sequences can be reduced to less redundant representatives through CD-hit.
2. Use the sequences to generate an SSN through all-by-all BLAST.
    a. The only requirement is that the BLAST output be in a tab separate format (such as outfmt 6), including the query accession, the subject accession and the bit score. Extra fields are fine, they are simply not used in the analysis.
3. Assemble annotations. At a minimum this will be each sequence's ID.

Subsequently Meta-SSN can be used to simplify and annotate a dense network:

1. The SSN is converted into a meta-network.
2. All sequences present in each meta-node are recorded.
    a. If representatives were used, then Meta-SSN can also extract all members under each representative if a CD-hit cluster definition file is provided.
3. Based on the members in each meta-node, Meta-SSN can be used to collect the related annotations (at the level of representatives and/or the full sequence set) and combine those using a number of summary methods.

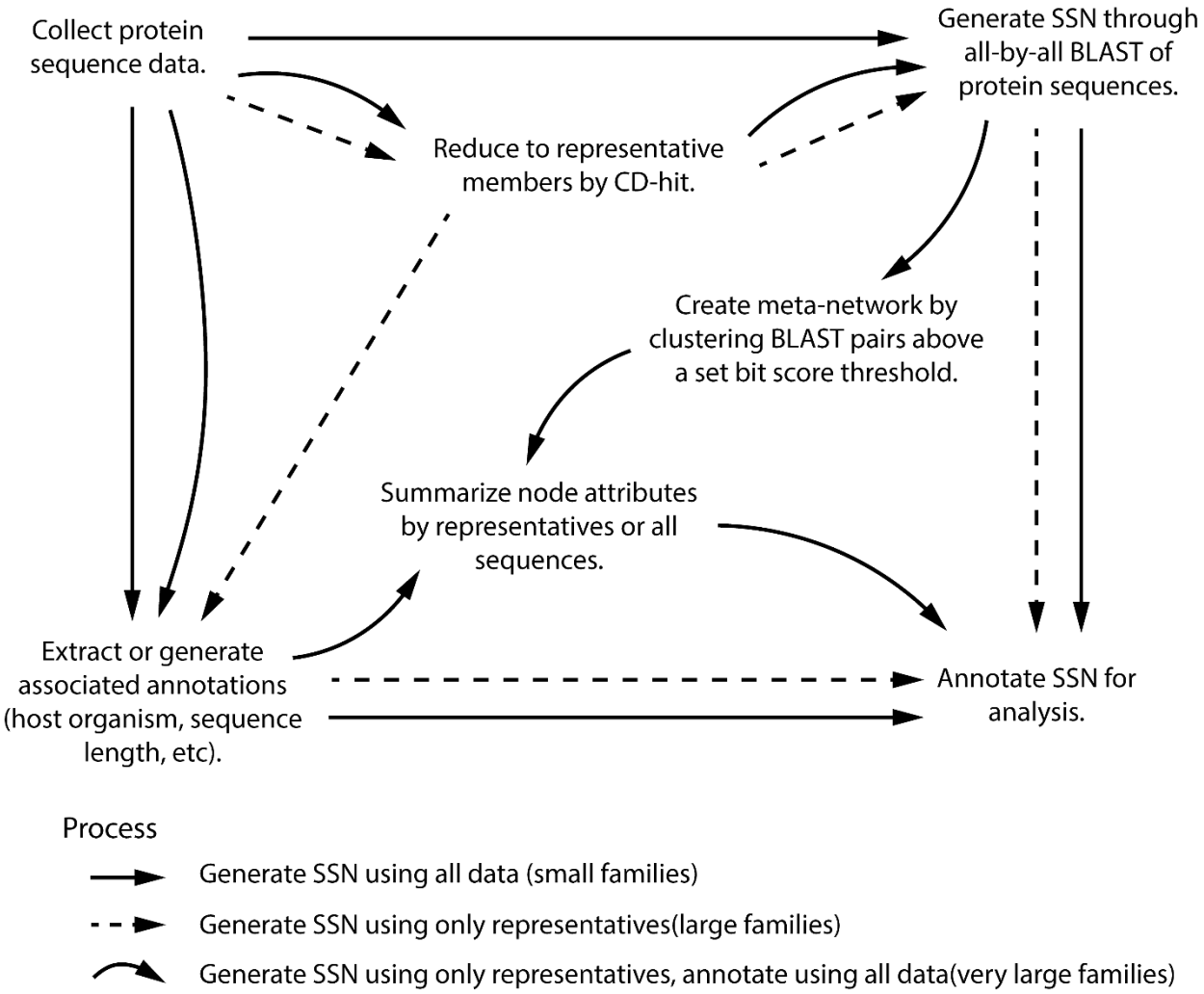

Figure 1. A comparison of common SSN workflows and the workflow of Meta-SSN. The four steps in the outer corner represent the minimal steps for SSN generation. Steps in the center represent various additional steps that reduce the density of the network for visualization.

2. Using Meta-SSN

2.1 Installation

Meta-SSN is comprised of a set of Python files, and can be run directly as long as an installation of Python version 3.7 or higher is present. There are currently no dependencies, including other Python packages. Make sure that the 'meta_network.py' script, the 'config.py' configuration file and the 'libs' folder are in the same directory when running the 'meta_network.py' script.

2.2 Running the program

Provided the configurations (see **Section 3** below) have been set, all processes are controlled by a single Python file 'meta_network.py'. Make sure that 'config.py' and 'meta_network.py' are in the same directory. Some options can be configured using the command line prompt. Type the following to bring up the help menu.

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

To generate a network that clusters sequences with BLAST bitscores above a ‘cluster_bitscore’ threshold into metanodes and keeps track of edges between metanodes above an ‘edge_bitscore’. Simply run the Python file using the following format from the command line.

```
python meta_network.py       edge_bitscore      cluster_bitscore
```

For example:

```
python meta_network.py      100      150
```

The above command would parse all the BLAST results provided to the program, and combine sequences into the same metanode if they have a bitscore of 150 or higher. Separate meta-nodes are connected if they have a bit score of at least 100 connecting any two members between the nodes; the BLAST hits with the highest bitscore above the threshold is used to represent the edge between the metanodes.

Determination of an appropriate cut-off for SSNs in general is a process of trial and error. Thus, the program also has options to generate a series of networks at different cut-offs. The first option is the ‘-sweep_interval’ setting (‘-si’ for short).

```
python meta_network.py      100      150      -si  25
```

By providing a sweep interval, the programs will generate a set of networks with increasing clustering thresholds starting at the given ‘edge_bitscore’ and ending at the given ‘cluster_bitscore’, keeping a difference in bitscore of ‘sweep_interval’ between the clustering threshold and edge threshold for each network. The above settings will produce networks with cut-offs of 75-100 (edge bitscore-cluster bitscore), 100-125 and 125-150.

Another way to generate a series of networks is to use the ‘-cluster_interval’ setting (‘-ci’ for short).

```
python meta_network.py     100      150      -ci   25
```

By supplying a cluster interval, networks with the same edge threshold and increasing clustering thresholds will be generated. The above command would generate networks at cut-offs of 100-125 and 100-150.

One thing to keep in mind is the number of BLAST results being stored when generating a metaSSN. A larger difference between the edge threshold and clustering thresholds increases the number of BLAST hits that need to be stored for the program for evaluating edges in the meta SSN, which will increase the memory usage of the MetaSSN script. For example, a network of 160-165 only has a bitscore difference of 5 between the clustering and edge thresholds, meaning only BLAST hits with a bitscore within that range need to be considered for constructing edges between metanodes; the program takes roughly 100-200mb of RAM when processing the BLAST data of 67million BLAST hits. In contrast, a network of 50-165 will keep track of all results above a bitscore of 50 and below 165, requiring up to 6gb of RAM when processing the same number of BLAST hits. If memory is somewhat limiting, try to use a low number for the ‘-sweep_interval’ setting (10 to 25), and avoid using the ‘-cluster_interval’ setting for a run with a large gap between the given ‘edge_bitscore’ and ‘cluster_bitscore’.

2.3 Additional options

Besides parameters for determining the cut-offs in SSN generation, the command line also allows for other options. There are two options related to file output.

- '-label' or '-l' – A string provided to this option will be used as the file prefix for all output files. If not given, the setting in the 'config.py' will be used.
- '-output_dir' or '-o' – A string provided to this option will be used as the directory to store all output. If not given, the setting in the 'config.py' will be used.

For example:

```
python meta_network.py    100    150    -l MBL    -o mbl_ssn
```

The above command would generate the network with cut-offs of 100-150. The output will be stored in a folder called 'mbl_ssn', and each file will have 'MBL' as the file prefix and the network file will be named 'MBL_100-150clust_network.txt' (see **Section 4** for file output formatting).

By default, the program relies on settings in the default configuration file (config.py). If the user wishes to store multiple different configurations side by side, those configuration files can be used in place of the default with the '-config_file' ('-cf' for short) option.

For example:

```
python meta_network.py    100    150    -cf  config2.py
```

The above command will use the configurations stored in the file 'config2.py' to conduct the analysis. As with the default configuration file, the file needs to be a python file ('.py' extension) and needs to be in the same directory as the script.

Furthermore, there are options to control the behavior of the program in terms of generating analysis. By default, metaSSN will conduct both declustering and annotations, though nothing will happen if no input is given for those options in the configuration file. If one has provided configurations but would like to not use them

- '-skip_declustering' or '-SD' – A flag. If this option is listed as part of the command, metaSSN will skip the step where the members of each metanode is used to extract all sequences from a set of CD-hit files.
- '-skip_annotation' or '-SA' – A flag. If this option is listed as part of the command, metaSSN will skip the step where information in the configuration file is used to annotate the SSN.

For example:

```
python meta_network.py    100    150    -SD   -SA
```

The above command would again generate the network with cut-offs of 100-150, but will skip both the declustering and annotation steps in MetaSSN. The output will just be a plain network file, and the only annotations for each metanode will be the number of member sequences in that metanode.

Finally, there are options to control the behavior of the program when results from previous analyses at the same cut-offs are found. By default, if MetaSSN detects a series of network and node files with the same name as the analysis currently being conducted, it will use these results, skipping the clustering step and going straight to the declustering and annotation steps. Similarily, if MetaSSN detects a declustered set of sequences with the exact filename of the current analysis, it will skip the declustering and use these old results. Both of these behaviors were designed so that the most time consuming steps of the analyses can be skipped if the user simply wishes to update the annotations. The settings below can disable these behaviors, forcing a new set of analyses.

- '-rebuild_networks' or '-R' – A flag. If this option is listed as part of the command, metaSSN will redo the analysis even if files from the same analysis have already been found.
- '-redo_declustering' or '-RD' – A flag. If this option is listed as part of the command, metaSSN will redo the declustering step even if an older file is already found.

For example:

```
python meta_network.py    100    150    -R   -RD
```

The above command will generate the network at cut-offs of 100-150, and redo all analyses even if an older set of files from analyses with the same cut-off are already found. This is useful for updating networks with newer data, or correcting older networks that have incorrect BLAST or CD-hit data. The other option is to simply delete the older files.

3. Meta-SSN configurations

The settings for Meta-SSN are contained within the 'config.py' file. All user adjustable settings are written in native Python. A small amount of Python knowledge is helpful, but templates are also provided for each setting.

The configurations are further split into 3 sub-sections, one for each of the major steps in the workflow.

3.1 Meta-network generation

These settings are used to create a meta-network from all-by-all BLAST data.

The following concern where to find the BLAST data, and how to read the BLAST file. All BLAST files should be using the same tab-delimited table format.

1. 'blast_results_path' : Provide a string indicating the path to the folder containing BLAST results. Do not include the individual file name.
2. 'blast_results_files': Optional. Provide a list of strings indicating which files in the 'blast_results_path' to use for meta-network generation. Set to None to use all files in the folder.
3. A series of numbers indicating which columns in the BLAST results correspond to which piece of information. Note that these are Python indices, such that the 1st position is 0, 2nd is 1, 3rd is 2, etc.
    a. 'query_col': Provide a number indicating which column in the BLAST file has the query accession.

b. ‘subject_col’: Provide a number indicating which column in the BLAST file has the subject accession.
c. ‘bitscore_col’: Provide a number indicating which column in the BLAST file has the bit score of the pairwise alignment.

The following settings can be set to only include BLAST hits from a subset of sequences for the meta-network.

4. Optional: These settings limit the sequences that are included in the meta-network. Providing these allows the user to generate a network targeting just a subset of sequences, which could speed up the process greatly if the base network is large.
    a. ‘filter_path’: The path to the folder containing filter files.
    b. ‘filter_files’: A list of files to use as filters. Provide an empty list to not use any filters. Filter files should be a plain text file with one sequence ID in each line.

The following settings concern how to manage the output.

5. ‘output_label’: A string indicating the filename to use for the result files of the network(s) generated. Other suffixes will be added to this label so keep it short and concise, for example simply noting the BLAST data source is enough.
6. ‘output_dir’: A string indicating where to save the output files.
    a. ‘use_old_metanodes’: A boolean (True or False) indicating if an old meta-network should be used if it is found in the given ‘output_dir’. Since meta-network generation can be time-consuming for large networks, this can save time if the user only intends to use subsequent Meta-SSN runs to add new annotations. Meta-networks will be considered the same only if the filenames are exactly the same as previously generated.
7. Optional: These settings can be applied if the user wishes to rename members in the network. For example, a sequence could be referenced by ‘sp|AX0PA’ or just ‘AX0PA’, depending on how the data was processed. This setting simply provides a convenience option so that later annotations can be more easily mapped. Names not found in the provided files are left as is.
    a. ‘mapping_path’: A string indicating the folder where the mapping file is stored.
    b. ‘member_name_mapping’: A list of strings indicating the file names of mapping files that should be used inside ‘mapping_path’. Mapping files should be a text file with a pair of sequence IDs separated by tab on each line, with the current name in the first column and the new desired name in the second.

3.2 Resolving sequence clustering (Optional)

By default the newly generated meta-network will keep track of all members that were combined in the base network. In the case that each of these members are also representatives of a similar set of sequences, Meta-SSN can associate all sequences clustered under a representative with the meta-node of the representative.

1. ‘cluster_info_path’: A string indicating the folder where the declustering files will be found.
2. ‘cluster_file_info’: A dictionary. Each key is the name of a file and the value is another dictionary with additional details on how to handle that file. In the inner dictionary, the following options are available for each file:

a. ‘format’: Set to ‘cdhit’ or ‘table’. If the format is set to ‘cdhit’, the file will be treated as a CD-hit cluster file, while ‘table’ will cause the file to be read as a table (each row has a representative sequence ID and a string of member sequence IDs). When setting to ‘cdhit’, none of the following options are required.
b. ‘delim’: For a table file, indicate the separator between table columns.
c. ‘key_col’: For a table file, indicate the index of the column that contains the ID of the representative sequence. In the table, this should be a single value.
d. ‘member_col’: For a table file, indicate the index of the column that contains the IDs of the member sequences. In the table, this should be a series of IDs separated by some separator such as ‘,’ or ‘;’.
e. ‘member_delim’: A character indicating the separator used for separating member IDs.

3. ‘cdhit_hierachy’: Optional. A dictionary. Each key is the name of a CD-hit filename provided in 3) and each value is a list of strings indicating CD-hit files that form a sequential set of cluster files with the file used as the key, in order of ascending % identity. This option is intended for declustering hierarchical CD-hit results where a set of sequences are first clustered at a target % identity cut-off, then the cluster representatives are used in a subsequent CD-hit with a lower % identity cut-off, and repeated until the lowest % identity cut-off.

3.3 Extra annotation (Optional)

If Meta-SSN is given attributes that can be matched to the members of each meta-node, the program can summarize these attributes for each meta-node in a number of ways. To make it easy to organize many different attributes, the input format for all different summary types are nearly identical. Meta-SSN expects a table format file, with each column separated by a character of the user’s choice. The minimum that an annotation file needs to include is the sequence ID, with other columns for other data. The same annotation file can also be used for all annotation options, so long as different attributes are separated into different columns. The same attribute can also be summarized using two or more of the different methods.

The settings below are all concerned with input for different annotation types, which will be briefly discussed first. The available annotation types are:

- Count (or sum): Count the number of sequences that belong to a meta-node, or add to a total if a column of numbers is given. Useful for counting sequences of a certain category within a node.
- Frequency: Given a column of discrete labels, count the number of each label in a meta-node then arrange them in order of frequency. Useful for comparing the proportion of different properties in a node.
- Label: Given a column of discrete labels, a meta-node will by tagged with each label if it is found at least once. Useful for indicating key members or properties, without repeating redundant labels.
- Distribution: Given a column of numbers, provide a distribution in the form of a 5 number summary. Similar to frequency, this is intended to summarize a property with a range of values but for continuous values rather than discrete.
- Membership in lower cut-off meta network: Given a meta-network (generated from the same BLAST data) at a lower bit score cut-off, find which meta-nodes in the current cut-off belong to

each node in the lower cut-off network. Useful for linking networks at different thresholds, since nodes from a higher clustering bit score all fragment from larger nodes at lower cut-offs.

All of the annotation types use a fixed input format, in which a list is provided that contains dictionaries. Each dictionary in the list describes the format of the table that stores the annotation data.

- 'files': A string or a list of strings that describe the filename of the input.
- 'id_col': A number indicating the index of the column in the file that contains the sequence IDs, which match those in the network.
- 'data_col': A number indicating the index of the column in the file that contains the annotation data.
- 'delim': The character that separates columns in the file.
- 'label': A string indicating how this annotation should be named in the node table.
- 'level': Either 'member' or 'sequence'. If set to 'member', annotations will only be assigned if the sequence ID is found as a direct member (representative) of the meta-nodes. If set to 'sequence', annotations will be assigned to any sequence that is a part of the node when declustered. Sequence level annotations will be ignored if no declustering was conducted.

An example of the input can be seen below.

gb|ADK25051.1|IND-12 [Chryseobacterium indologenes] 109 IND B1

gb|CAJ32373.2|IND-6 [Chryseobacterium indologenes] 110 IND B1

gb|AAG29757.1|IND-2 [Chryseobacterium indologenes] 111 IND B1

gb|ADK25050.1|IND-11 [Chryseobacterium indologenes] 112 IND B1

gb|BAJ14288.1|IND-15 [Chryseobacterium indologenes] 113 IND B1

gb|CAA29819.1|BcI [Bacillus cereus] 114 BcI B1

gb|AAA22562.1|BcII [Bacillus cereus] 115 BcII B1

gb|CAD37801.1|SPM-1 [Pseudomonas aeruginosa] 116 SPM B1

gb|AVX51087.1|CAM-1 [Pseudomonas aeruginosa] 117 CAM B1

gb|AIY26289.1|GIM-2 [Enterobacter cloacae] 118 GIM B1

gb|CAF05908.1|GIM-1 [Pseudomonas aeruginosa] 119 GIM B1

gb|AAT90846.1|SLB-1 [Shewanella livingstonensis] 120 SLB B1

gb|AAT90847.1|SFB-1 [Shewanella frigidimarina] 121 SFB B1

gb|AAN63647.1|MUS-1 beta-lactamase [Myroides odoratimimus] 122 MUS B1

The table has 4 columns separated by the '\t' character, with the full sequence header in the 1st column, the sequence index (sequence header converted to consecutive numbers) in the 2nd, the protein name in the 3rd and the protein family type in the 4th. This table could be described as the following dictionary

when trying to use the protein name for the ‘label’ method. Note that column numbers are converted to index, so that column 2 (‘id_col’) is 1 and column 3 (‘data_col’) is 2.

{'files':'b1_name.txt','id_col':1,'data_col':2 ,'delim':'\t', 'label':'known B1 families', 'level':'sequence'}

Some of the annotation types also have some specialized settings for the dictionary.

- Count: The ‘data_col’ can be set to a column or None. When set to None, the number of sequences found in a meta-node is simply tallied. When set to a numerical column, the numbers in each meta-node will be summed.
- Frequency: An additional setting ‘highest_entries’ needs to be set to a number or None. The setting restricts the number of properties with the highest frequency to display. If set to 3, the 3 highest frequency properties will be displayed as in full while all lower frequency properties are combined into a the ‘other’ category. If set to None, all properties will be shown as individual frequencies.
- Membership in lower cut-off network: An additional setting ‘bitscore’ needs to be set to the bit score cut-off of the lower cut-off network. This is a safety feature so that the annotation is not applied when the current meta-network being annotated is a lower cut-off that the input network used for annotation, which would lead to non-sensical results.

4. Meta-SSN output

Meta-SSN provides 4 pieces of output, two of which are part of the SSN while the other two provide additional information about meta-node membership. The files are saved in the location specified by ‘output_dir’. Different output file types can be distinguished by the suffix, which is attached to the ‘output_label’ and clustering setting given earlier. The filename structure is shown below.

‘<output_label>_<edge bitscore>-<cluster bitscore>clust_<suffix>.txt’

The following files are part of the SSN:

1. ‘network’ suffix: This file is the network file, which is the file with the connectivity information sufficient to generate the SSN. The output is in the following format.

node1	node2	bitscore	id
1	2	164.0	0
1	27	161.0	1
1	99	164.0	2

   Each row is an edge in the network, where ‘node1’ and ‘node2’ are the connected meta-nodes and ‘bitscore’ is the edge attribute. ‘id’ is a unique ‘id’ for the edge and is not usually needed for anything, as most network visualization software will generate their own. To load the file into a network visualization tool, the file should be loaded as a network file and the ‘node1’ and ‘node2’ columns should be given as the source and target nodes and edges should be set to undirected.
2. ‘node_summary’ suffix: This file is the node attributes file, which provides additional information on each meta-node in the network. Each row in the output represents a meta-node, as noted in the ‘node’ column. Each additional column is an annotation that was added during

the annotation step, with 'member_count' (the number of members sequences in the meta-node) provided by default. 'seq_count' (the number of declustered sequences in the meta-node) are also provided by default if declustering was conducted. To load the file into a network visualization tool, the file should be loaded as the node table with the 'node' column as the key column.

The following files provide information on the exact sequences in each meta-node. These

3. 'membership' suffix: This file lists the sequences that are direct members of each meta-node (no declustering). The 1st column indicates the node while the 2nd column indicates the sequence ID.
4. 'node_all_seqs' suffix: This file lists all sequneces that could be assigned to each meta-node after declustering. Similar to the 'membership' file, 1st column indicates the node while the 2nd column indicates the sequence ID. This file is not generated if no declustering was conducted.

5. Other tools

5.1 Subnetwork extraction

When SSNs are very large, it could become difficult to annotate and explore as there many more nodes and edges than what the user is interested in. To overcome this issue, the user could opt to only examine nodes with a property of interest and their 1st step neighbours. The 'fetch_subnetwork.py' script aims to provide this functionality. The user provides a set of meta-nodes that they are interested in for a given meta-network, and the script can find all meta-nodes in networks at other clustering bit scores that contain any members of the given meta-nodes of interest. For example, if node 136 is given as the input from a meta-network at a clustering bit score of 150, the members in that node can be used to search for all nodes that contain these members at networks of any clustering bitscore (120, 250, 180, etc.). Similar to 'meta_network.py', the 'fetch_subnetwork.py' script only requires an installation of Python 3 and can be run directly from command line.

```
Python fetch_subnetwork.py
```

There are 4 main inputs for the 'fetch_subnetwork.py':

1. 'file_template': A string indicating the file name structure to look for. Only the part before the first set of '{}' brackets needs to be adjusted, and should be set to the 'output_label' of the desired network.
2. 'results_dir': A string indicating the folder where the meta-networks needed for extraction are located. The meta-networks used as targets and meta-networks from which subnetworks should be extracted should be in the same folder.
3. 'base_node_members': A dictionary. Each entry indicates a set of nodes that the user wishes to extract, with the key being the label to assign for extracted nodes and the value being another dictionary that specifies the source of the node. Targets from multiple cutoffs, or different nodes from the same cutoff can be used in tandem, so long as the labels (dictionary key) are unique. The inner dictionary has the following settings:
    a. 'cutoff': A tuple of numbers indicating the edge and clustering bitscores of the meta-network that contains the target nodes.
    b. 'nodes': A list of numbers indicating the meta-nodes that should be used as targets.

4. ‘network_cutoff’: A list containing tuples of numbers. Each tuple indicates the the edge and clustering bitscores of a meta-network that a subnetwork will be extracted from.
5. ‘outdir’: A string indicating the folder to place the results of this script.

There are 2 output files, with similar naming structure to the main output of Meta-SSN. The type of the file is described by the suffix.

‘<output_label>_<edge bitscore>-<cluster bitscore>clust_<suffix>.txt’

1. ‘subnetwork’ suffix: This is the extracted network file containing only nodes with members from target nodes and the 1st step neighbours.
2. ‘node_info’ suffix: This is a file containing label for each extracted meta-node based on which members belong to each of the target meta-nodes. Neighbour nodes that are not found in targets are not labelled. The ‘node’ column indicates the meta-node while the ‘node_assoc’ column indicates the target meta-node labels.