Contents

1 Running a Neo4j database **2**
- 1.1 Neo4j Docker . . . 2
- 1.2 Neo4j on a local machine . . . 3
  - 1.2.1 Neo4j as a service on a Linux system . . . 3
  - 1.2.2 Neo4j as a console application on a macOS system . . . 4
  - 1.2.3 Neo4j as a console application on a Windows system . . . 4
- 1.3 Neo4j Browser . . . 4
- 1.4 Neo4j server . . . 5

2 Cypher basics for biologists **6**
- 2.1 Introduction . . . 6
- 2.2 *Pisaster* network . . . 6
- 2.3 Database schemas . . . 6
- 2.4 Connecting to a Neo4j database . . . 8
- 2.5 Creating nodes and relationships . . . 8
- 2.6 Querying the database . . . 10

3 A database schema for microbial association networks **12**
- 3.1 Introduction . . . 12
- 3.2 Database schema overview . . . 12
- 3.3 Default nodes and relationships . . . 13

4 mako - vignette **16**
- 4.1 Accessing a Neo4j database . . . 17
- 4.2 Uploading the BIOM file . . . 17
- 4.3 Uploading two network files . . . 17
- 4.4 Extracting the overlap between two networks . . . 19
- 4.5 Calculate metadata correlations . . . 19
- 4.6 Merge network to a higher taxonomic level . . . 20
- 4.7 Export a network to Cytoscape . . . 20

5 mako - demo file with 60 networks **23**

6 mako - manual **24**
- 6.1 Introduction . . . 24
- 6.2 CLI . . . 24
  - 6.2.1 mako base . . . 25
  - 6.2.2 mako neo4biom . . . 25
  - 6.2.3 mako io . . . 26
  - 6.2.4 mako netstats . . . 26
  - 6.2.5 mako metastats . . . 27
  - 6.2.6 mako anuran and manta . . . 27
- 6.3 GUI . . . 27
- 6.4 API . . . 27

1 Running a Neo4j database

To use mako to store biological data, it needs to be able to access a Neo4j database. There are multiple ways to run one; in this guide, several strategies for using Neo4j will be addressed, with relevant links to Neo4j sources included.

Two editions of Neo4j are available: Neo4j Community and Neo4j Enterprise. Neo4j Community is open source and freely available, while Neo4j Enterprise requires a licence. Many of the advanced features available to Enterprise are not necessary for simpler biology applications, and therefore will not be discussed in this guide. For a full overview of features available to Neo4j Community and Neo4j Enterprise editions, please take a look at the Neo4j web page.

In this guide, three options for accessing Neo4j database will be discussed: via Docker, via Neo4j Desktop and via a server.

1.1 Neo4j Docker

Docker is a convenient tool for virtualization. Docker containers contain a standardized environment that can be run from any desktop or via the cloud, solving many issues with compatibility across operating systems. By running software via Docker instead of directly on a desktop, it becomes possible to run different versions of dependencies side-by-side without affecting existing installations. As a result, running Neo4j via Docker can be more convenient. By default, encryption is not supported by the Neo4j container. This can be important when connecting to the database with Neo4j drivers (e.g. those included in mako).

For most users, it will be simplest to run Docker Desktop on a local machine. Mac and Windows users need to use Docker Desktop, while Linux users can run Docker containers after installing docker-engine. Please make sure Docker can run on your machine using one of the above links. Mac and Windows users also need to launch Docker Desktop after installation to start using Docker containers.

After installing Docker (Desktop), the Neo4j Docker container can be setup. Each time you call the Docker container for Neo4j, a Neo4j Docker image is pulled from DockerHub and used to start a container. You can specify a range of commands to configure the Neo4j Docker image. For an expansive guide, please take a look at the Neo4j Docker how-to.

To have a clean testing environment for integration tests, mako uses a newly-started Docker container that can be interacted with. This container is configured so it does not conflict with a default instance of Neo4j Desktop. The command can also be used to run the Neo4j container outside mako's testing environment. After installing Docker and starting Docker Desktop (if applicable), the mako command can be used to start the Neo4j container:

```
docker run \
    --rm \
    -d \
    --publish=7475:7474 --publish=7688:7687 \
    --name=neo4j \
    --env NEO4J_AUTH=neo4j/test \
    neo4j:latest
```

The –**rm** flag tells Docker to clean up after exiting, while the -d flag means the Docker can be accessed separately. Importantly, the –**publish** flag changes exposed ports, so the Neo4j container can be accessed via navigating to http://localhost:7475/browser/ in a browser and setting the Bolt connection to bolt://localhost:7688. The –**name** flag sets the container's name, the –**env** flag sets the username and password for the Neo4j container and **neo4j:latest** tells Docker to pull the latest image. To stop the Docker container is then quite straightforward:

```
docker stop neo4j
```

To start using the Neo4j Docker container, simply run the first command or a modified version of this command, and then navigate to Neo4j Browser to access the database. By passing these parameters to mako, you can have mako import BIOM files and networks to the database. Keep in mind that by default, the data folder is cleared once the container is shut down, so Docker is not as useful for maintaining a local database.

1.2 Neo4j on a local machine

For starting out with Neo4j, a local Neo4j database can also be helpful. You can download Neo4j from the Neo4j Downloads page by selecting the appropriate edition listed under Community Server. For more extensive analyses, maintaining a local Neo4j database can be helpful since the database is persistent.

To use Neo4j, you first need to follow the installation guide. The Neo4j website has extensive instructions for Linux, macOS and Windows in the Neo4j Operations Manual. For all operating systems, Neo4j can be run as a service that can then be accessed via Neo4j Browser. However, the exact details of setting up the server are different per system. The instructions below are simplified versions of some of the Neo4j instructions.

1.2.1 Neo4j as a service on a Linux system

You can find all instructions for running Neo4j as a Linux service in the Linux section of the Operations Manual. To run Neo4j on Linux, you need to first install the RPM package. Instructions for installations of the RPM package can also be found in the Linux section of the Operations Manual. After installation of RPM (and possibly rebooting your system), you can follow the instructions below to start the service.

- Run **systemctl start neo4j** to start the service
- Run **systemctl stop neo4j** to stop the service

1.2.2 Neo4j as a console application on a macOS system

You can find all instructions for running Neo4j as a macOS service in the macOS section of the Operations Manual. While you can run Neo4j as a console application, you can also run it as a service. However, you need to create the service manually using launchd.

- Download the macOS version of Neo4j listed under Community server
- Unzip the Neo4j download in a folder of your choice
- In command line, navigate to the Neo4j folder where you unzipped the download
- Run **./bin/neo4j console** to start the server
- Run **CTRL+C** to stop the server

1.2.3 Neo4j as a console application on a Windows system

You can find all instructions for running Neo4j as a Windows service in the Windows section of the Operations Manual.

- Download the Windows version of Neo4j listed under Community server
- Unzip the Neo4j download in a folder of your choice
- In command line, navigate to the Neo4j folder where you unzipped the download
- Run **bin\neo4j console** to start the server
- Run **CTRL+C** to stop the server

1.3 Neo4j Browser

Neo4j Browser is a user interface for Neo4j that can be accessed via a browser. After a Neo4j service or console application is running, the Neo4j Browser can be started by navigating to http://localhost:7474/ (or another port if port settings are adjusted). Neo4j Browser provides a place to enter Cypher queries and visualize simple results. You can find more details on Neo4 Browser on the Neo4j Browser section of the developer guides.

1.4 Neo4j server

For hosting a persistent Neo4j database that is accessible to multiple users and can be encrypted, it may be beneficial to host Neo4j on a server and configure it appropriately. Since the Community Edition only supports a single database, this is less suitable for extensive server solutions. The Neo4j developer guides contain additional documentation on Neo4j administration for production. Guides for deploying Neo4j in the cloud can also be found in the cloud section of the operations manual.

2 Cypher basics for biologists

2.1 Introduction

Neo4j is a native graph database, meaning it can store biological data in a way that is more intuitive to people unfamiliar with SQL. In this guide, we will use Paine's food web [2] to demonstrate how database schemas work and how you can use Cypher queries to access a Neo4j database. We will use the Neo4j Browser to visualize query outcomes. All code in this guide will be in pure Cypher, meaning you can run it from the browser interface.

The Cypher queries in this file can be used to recreate Paine's network in your own Neo4j database. If you want to know how to run Neo4j locally, connect to a server or connect to the Docker container provided with this guide, we refer to the instructions described in: Running Neo4j: the basics.

2.2 *Pisaster* network

Paine studied food webs of rocky shores in North America. In particular, he was interested in predator-prey interactions and carried out removal experiments to assess how species removal affected community structure. One species in particular appeared to have an outsized effect on the community - *Pisaster ochraceus*, a carnivorous starfish. In the subweb containing *Pisaster* and its food items, the numbers represent the fraction of food items in terms of number of food items and in terms of consumed calories (Figure 1). We will demonstrate the flexibility of Neo4j by discussing several strategies for writing Paine's subweb to a Neo4j database.

2.3 Database schemas

Neo4j does not enforce constraints, it is possible to store information in any way possible. The Neo4j introduction to graph database concepts provides some details on how this happens. To facilitate straightforward methods to access information, it can be helpful to come up with a set of rules for how data should be stored. By storing data according to a set of rules, it becomes easier to define general Cypher queries that can access the appropriate data.

When deciding on how the data is stored, the result is usually a database schema. Schemas can be visualized as graphs, but they can also be described verbally as a set of rules or constraints. A database schema contains nodes (pieces of information) and relationships between these nodes. Nodes can have properties. For example, if we want to make a node that represents *Pisaster*, we could choose to have a node with the label **Starfish**, a **Genus** property that says *Pisaster* and a **Species** property that says *ochraceus*. However, for speed of access of databases it can matter a lot whether information is stored as a (node or relationship) label or as a separate node. This is because the Neo4j

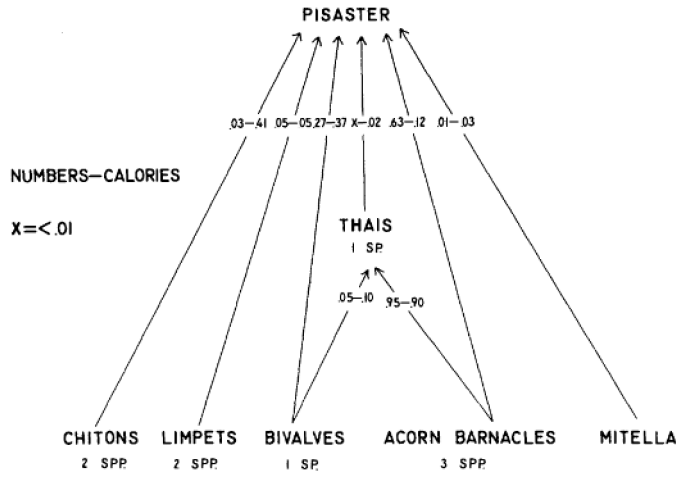

Figure 1: Food web showing *Pisaster* and the species Paine saw it eat.

databasement management system (DBMS) uses graph traversal to return query results. Moreover, we should also keep in mind that Paine chose to keep his food web illustrations simple, so a single node represents an entire group of organisms.

If we were to define *Pisaster* as described above, a Cypher query to find all species belonging to the genus *Pisaster* would look like this:

```
MATCH (n:Organism {Genus: 'Pisaster'}) RETURN n
```

The DBMS needs to access the property Genus. For a simple query like this, the speed loss of accessing properties is not that large, but for complicated queries and larger databases, this can become important. Therefore, if we have a large enough database that we might have several species of *Pisaster* and need to query by genera, it makes sense to add a separate node with **Genus** label. The query to find all instances of *Pisaster* would then look more like this:

```
MATCH (n:Organism)–(:Genus {name: 'Pisaster'}) RETURN n
```

In addition to speed advantages, the Genus node only needs to exist once in the database and can then be connected to all other instances of *Pisaster*. Again, for small networks this is not a significant problem, but if we have a database with 100000 instances of the phylum Proteobacteria, we might start getting performance gains by storing Proteobacteria as a single separate node rather than a node property for each taxon, OTU or ASV.

For this example, two different database schemas could make sense for the *Pisaster* subweb (Figure 2). We could also include each organism as a Predator or Prey node, but that would not make a lot of sense for *Thais* sp., since these organisms are both predator and prey. However, many other variations are possible, all of which contain the exact same information but may have consequences for query speed and query complexity. One schema is not necessarily better than the other; rather, it depends on the needs to use this information. The fraction can be considered both a separate node and a property of the predator-prey relationship. Both options are fine and neither is expected to affect performance of this subweb, since accessing relationship properties is unlikely to take a prohibitively long time. For this demo, we will use the schema with fractions as relationship properties, to address how these can be used to query a network.

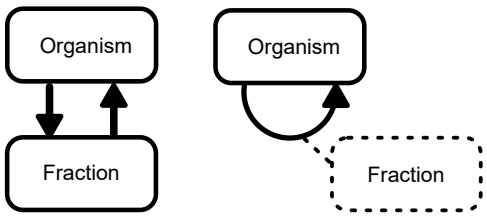

Figure 2: Schemas for storing a food web.

2.4 Connecting to a Neo4j database

We can use the Neo4j Docker container to run Neo4j without needing to deal with a complicated setup. For this demo, the following command was used to start the Docker container:

docker run –rm -d –publish=7475:7474 –publish=7688:7687 –name=neo4j –env NEO4J_AUTH=neo4j/test neo4j:latest

We can navigate to the Neo4j Browser via the specified port: **http://localhost:7475/browser/**. To connect to Neo4j, we then fill in the rest of the information specified in the command; note that NEO4J_AUTH contains the username and password for the Neo4j database in the Docker container (Fig 3).

2.5 Creating nodes and relationships

After connecting to the Neo4j database, the textbox at the top of the browser can be used to type Cypher queries and interact with the database. For convenience,

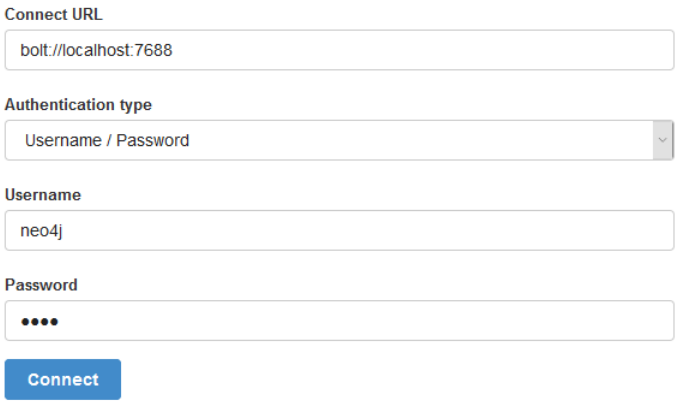

Figure 3: Interface of Neo4j Browser with bolt://localhost:7688 as connecting URL and username / password authentication.

we will unwind a list of organism names (first 2 commands) and then use the MERGE query in combination with the unwound values to create all organisms. A Cypher query contains several clauses, MERGE being one of them. Each clause tells the DBMS what to do. In this case, the clauses tell the DBMS to unwind the list and match or create a node. Those nodes are then returned.

- WITH ['Pisaster', 'Thais sp.', 'Chitons', 'Limpets', 'Bivalves', 'Acorn barnacles', 'Mitella'] as organisms
- UNWIND organisms as x
- WITH x
- MERGE (a:Organism {name: x}) RETURN a

The Neo4j Browser will automatically highlight the structure of the query (Figure 4). Additionally, the query returns all created nodes. You should see the created nodes as disconnected, floating circles in the browser.

```
1 WITH ['Pisaster','Thais sp.','Chitons','Limpets','Bivalves','Acorn
  barnacles'] as organisms
2 UNWIND organisms as x WITH x
3 MERGE (a:Organism{name:  x}) RETURN a
```

Figure 4: Neo4j Browser query.

Next, we create the relationships between the nodes. Because this is slightly more complicated and includes setting relationship properties, we will run a separate query for each relationship. Each query first finds the two organisms

(MATCH) and then creates the relationship with number and calories values (MERGE). Again, these are returned if successful.

- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Chitons'}) MERGE (a)-[r:PREYS_ON {number: 0.03, calories: 0.41}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Limpets'}) MERGE (a)-[r:PREYS_ON {number: 0.05, calories: 0.05}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Bivalves'}) MERGE (a)-[r:PREYS_ON {number: 0.27, calories: 0.37}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Thais sp.'}) MERGE (a)-[r:PREYS_ON {number: 'x', calories: 0.02}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Acorn barnacles'}) MERGE (a)-[r:PREYS_ON {number: 0.63, calories: 0.12}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Pisaster'}), (b:Organism {name: 'Mitella'}) MERGE (a)-[r:PREYS_ON {number: 0.01, calories: 0.03}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Thais sp.'}), (b:Organism {name: 'Bivalves'}) MERGE (a)-[r:PREYS_ON {number: 0.05, calories: 0.10}]-(b) RETURN a, b
- MATCH (a:Organism {name: 'Thais sp.'}), (b:Organism {name: 'Acorn barnacles'}) MERGE (a)-[r:PREYS_ON {number: 0.95, calories: 0.90}]-(b) RETURN a, b

2.6 Querying the database

To get the complete network, we can run the query below. Notice the LIMIT statement; when we have networks with thousands of nodes, it is crucial to limit the number of returned items to prevent the Neo4j Browser from crashing. In this case, the LIMIT is higher than the total number of items, so we simply get the complete network.

```
MATCH (n) RETURN n LIMIT 50
```

You should get a graph in the Neo4j Browser that contains the entire food web. When you mouse over the relationships, you can see the values; when you drag and drop nodes, you can even rearrange the entire network so it looks like Figure 1! You can see an example of the Neo4j network below (Figure 5.

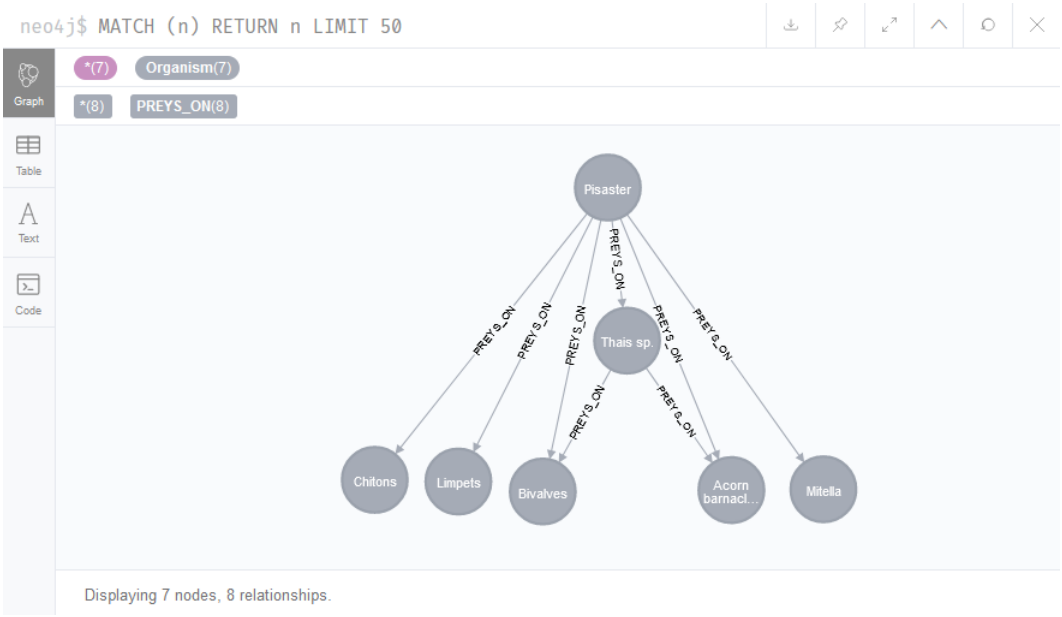

Figure 5: Neo4j Browser query.

Since the relationships are directed, there is not even truly a need to specify that some species are predators and others are prey. We can simply use the directionality to find species that interact in a certain direction. The first of these queries returns all predators, the second returns all prey! Finally, the third query filters on the relationships so that only prey are returned that make up over 20% of a predator's calories.

- MATCH (a)->(b) RETURN a
- MATCH (a)–>(b) RETURN b
- MATCH (a)-[r]->(b) WHERE r.calories > 0.20 RETURN b LIMIT 50

Explaining the full potential of Cypher is not within the scope of this short guide. Note that many other clauses have not been addressed here, such as CREATE. For a complete overview of Cypher, we refer to the Neo4j manual.

3 A database schema for microbial association networks

3.1 Introduction

Database schemas are blueprints that describe how data in a database is organized. By defining schemas, we are able to design queries that can access data stored according to those schemas. Hence, mako's core component is a database schema constructed from several OWL terms. This database schema also defines constraints for the data and provides guidance on how data needs to be uploaded to become easily accessible. Most functionality of the API uses the schema to rapidly upload BIOM files, check edge weights or find associations between related species.

To properly integrate with Neo4j's ability to work with ontology web language (OWL), each node and relationship label is derived from an existing ontology, the NCI Thesaurus [3]. However, since relationships are not well-defined across biological ontologies, constraints for relationships were defined manually with Protégé. For each relationship, domains and ranges (target and source nodes) were specified. While these axioms are not added to Neo4j as actual constraints, mako can run checks to see if domain and range axioms are violated by any nodes or relationships in the database. [1].

In this chapter, the database schema used by mako is described in detail to aid in the design of custom queries that can carry out additional tasks on Neo4j databases.

3.2 Database schema overview

The database schema used by mako was designed to meet several demands:

- Simple querying of BIOM data
- Simple querying of network data
- Respect hierarchical structure of taxonomic tree
- Straightforward access of networks for meta-analyses
- Flexible inclusion of metadata

As a result, each of the items always present in networks and BIOM files have their own node label and can more easily be accessed. In contrast, property nodes are more flexible and can be connected to any other node. The complete schema, sufficient to import BIOM files and network files, therefore specifies both standard node types and node relationships (Figure 6).

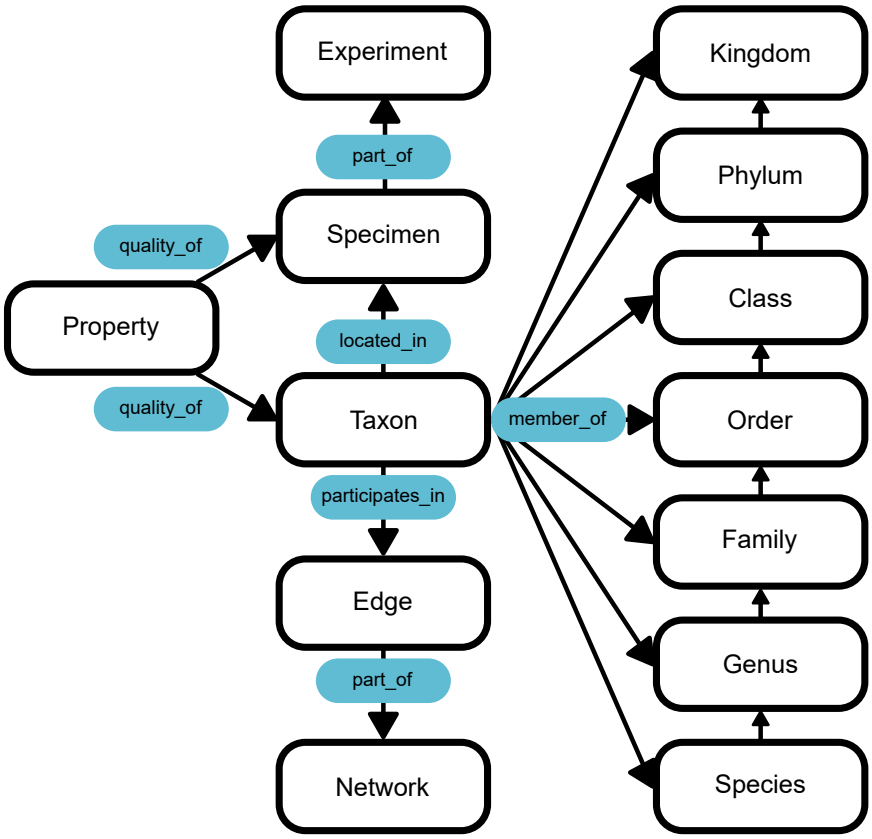

Figure 6: Schema for storing BIOM files and microbial networks.

3.3 Default nodes and relationships

Many of the nodes and relationships have properties that represent specific characteristics. The overview below contains a description of each node or relationship, its ontology term and any properties that the item has if default import functions are used.

Edge - NCIT:C75923

The Edge node is one of the most important nodes of mako and supports most of its functionality. While it is possible to connect Taxon nodes directly, the Edge node can be connected separately to Network nodes or Property nodes, so that more complicated Edge metadata is possible. For example, Edges could be connected to a Network node named Literature if they are identified via

a literature analysis. Hence, this implementation expressly facilitates network meta-analyses.

- **name** Edge nodes normally are assigned a universally unique identifier (UUID) as a name property, since edge lists and matrix formats do not contain edge identifiers.
- **Weight** If available, Edge nodes are also assigned a weight property. Since edge weights are usually not directly comparable across network inference methods, it is recommended to use a binary representation of 1 and -1 for positively- and negatively- weighted edges.

Experiment - NCIT:C42790

The Experiment node connects all Specimen nodes to a single file.

- **name** The name for the Experiment node is taken from the filename of the imported file, but it can also be adjusted manually.

Network - NCIT:C61377

The Network node connects all Edge nodes to a single network. An alternative version of Network nodes is also available; these have the Set label and are used to indicate results of network operations.

- **name** The name for the Network node is taken from the filename of the imported file, but it can also be adjusted manually.

Property - NCIT:C20189

Property nodes are nodes that can be used to query specific metadata. These can be connected to any other node, but default imports connect them to Specimen and Taxon nodes.

- **name** The name for the Property node is taken from the column name (if table-like import). For example, a column with pH values will lead to one pH property being generated.

Specimen - NCIT:C19157

Specimen nodes represent specimens or samples collected for an experiment.

- **name** The name for the Specimen node is taken from whatever sample identifier is used by the file, e.g. column names for standard count tables.

Taxon - NCIT:C40098

Taxon nodes usually represent OTUs or ASVs, but can also represent any other biological unit.

- **name** The name for the Taxon node is taken from whatever taxon identifier is used by the file, e.g. row names for standard count tables.

Species, Genus, Family, Order, Class, Phylum, Kingdom

NCIT:C45293, NCIT:C45292, NCIT:C45290, NCIT:C45287, NCIT:C45280, NCIT:C45277, NCIT:C45276
These nodes represent taxonomic values for these specific taxonomic levels. Taxon nodes usually represent OTUs or ASVs, but can also represent any other biological unit.

- **name** The name for taxonomy nodes is the value usually represented in taxonomy tables. For example, the Genus node for *Escherichia coli* will have *Escherichia* as a name. When tables contain only a prefix (e.g. g__), no node is created.

LOCATED_IN

Relationships connecting Taxon nodes to Specimen nodes.

- **count** Value of count or abundance table for a specific taxon in a specific specimen.

MEMBER_OF

Relationships connecting Taxon nodes to taxonomy nodes (Species, Genus, Family, Order, Class, Phylum and Kingdom), and also taxonomy nodes to each other. These relationships form acyclic trees with Kingdom as the root and therefore mimic taxonomic trees. By directly connecting Taxon nodes to taxonomy nodes, queries can access those directly.

PARTICIPATES_IN

Relationships connecting Taxon nodes to Edge nodes. These relationships are currently undirected, so Taxon nodes always participate in an edge, rather than being affected by another taxon.

PARTICIPATES_IN

Relationships connecting Sample nodes to Experiment nodes and Edge nodes to Network nodes. These relationships therefore indicate a hierarchy of file structures.

QUALITY_OF

Relationships connecting nodes to Property nodes. Usually, these relationships connect Taxon and Specimen nodes to Property nodes, but other nodes can be connected to Property nodes as well.

- **value** Value of the Property node. For example, if pH values per sample are uploaded to the database, one pH Property node is generated and all QUALITY_OF relationships contain the pH value for that specific Specimen node.

4 mako - vignette

In this vignette, several of mako's functions will briefly be introduced with an example data consisting of only 5 taxa and 20 samples. This data set is simple enough that most relevant nodes can be visualized in Neo4j Browser, making it feasible to follow along with all of mako's operations.

The tables below give an overview of the data contained in the BIOM file. The file and associated networks can be downloaded from the mako Github repository.

Count table:

OTU	Sample1	Sample2	...	Sample20
OTU_1	0	0	...	6
OTU_2	5	1	...	1
OTU_3	0	0	...	0
OTU_4	2	1	...	0
OTU_5	0	1	...	0

Taxonomy table part 1 (only OTU_3 is Archaea):

OTU	Phylum	Class	Order
OTU_1	p__Proteobacteria	c__Gammaproteobacteria	o__Enterobacteriales
OTU_2	p__Firmicutes	c__Clostridia	o__Halanaerobiales
OTU_3	p__Euryarchaeota	c__Methanomicrobia	o__Methanosarcinales
OTU_4	p__Firmicutes	c__Clostridia	o__Halanaerobiales
OTU_5	p__Proteobacteria	c__Gammaproteobacteria	o__Enterobacteriales

Taxonomy table part 2:

OTU	Family	Genus	Species
OTU_1	f__Enterobacteriace	g__Escherichia	s__
OTU_2	f__Halobacteroidaceae	g__Sporohalobacter	s__lortetii
OTU_3	f__Methanosarcinaceae	g__Methanosarcina	s__
OTU_4	f__Halanaerobiaceae	g__Halanaerobium	s__Halanaerobium saccharolyticum
OTU_5	f__Enterobacteriace	g__Escherichia	s__

Sample metadata (except BarcodeSequence and LinkerPrimerSequence):

Sample	BODY_SITE	Description
Sample1-10	gut	human gut
Sample11-20	skin	human skin

Edge list:

Network	Target	Source	Weight
demo_1	OTU_1	OTU_2	1
demo_1	OTU_2	OTU_5	1
demo_1	OTU_3	OTU_4	-1
demo_1	OTU_1	OTU_5	1
demo_2	OTU_1	OTU_2	1
demo_2	OTU_2	OTU_5	1
demo_2	OTU_1	OTU_5	-1

4.1 Accessing a Neo4j database

For running a Neo4j database, please refer to Chapter 1: Running a Neo4j database. There needs to be an online database for mako to access before any of the following commands will work. The software will connect to the running database and interact with this. For this tutorial, it is assumed that the settings to connect to Neo4j are identical to those used for the Docker instance.

4.2 Uploading the BIOM file

Navigate to the location where you downloaded the BIOM file. By storing the config file, we can save ourselves the hassle of typing access information each time. For **-fp**, fill in the file path where the downloaded BIOM file is stored.

```
mako neo4biom -fp local\_filepath -cf -u neo4j -p test
-a bolt://localhost:7688 -biom demo.biom
```

If you access the Neo4j Browser (http://localhost:7475/browser/) and run the following query, you should be able to access all the nodes connected to taxa:

```
MATCH p=(n:Taxon)--() RETURN p LIMIT 50
```

4.3 Uploading two network files

Navigate to the location where you downloaded the network files. Now, only the network files and the file path need to be provided.

```
mako io -fp local\_filepath -cf -net demo_1.graphml demo_2.graphml
```

If you access the Neo4j Browser (http://localhost:7475/browser/) and run the following query, you should be able to access all the nodes connected to edges:

```
MATCH p=(n:Edge)--() RETURN p LIMIT 50
```

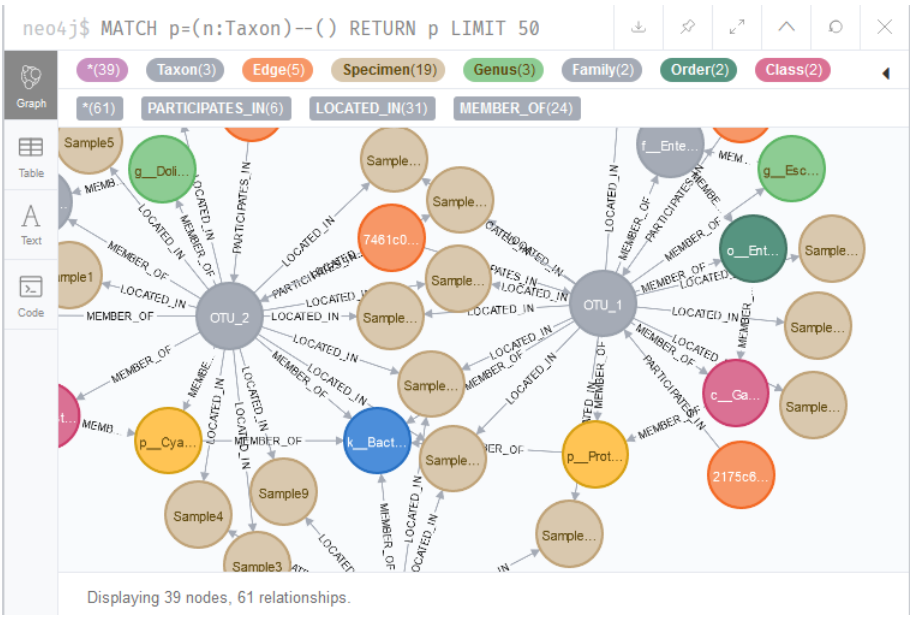

Figure 7: Taxon links to other nodes.

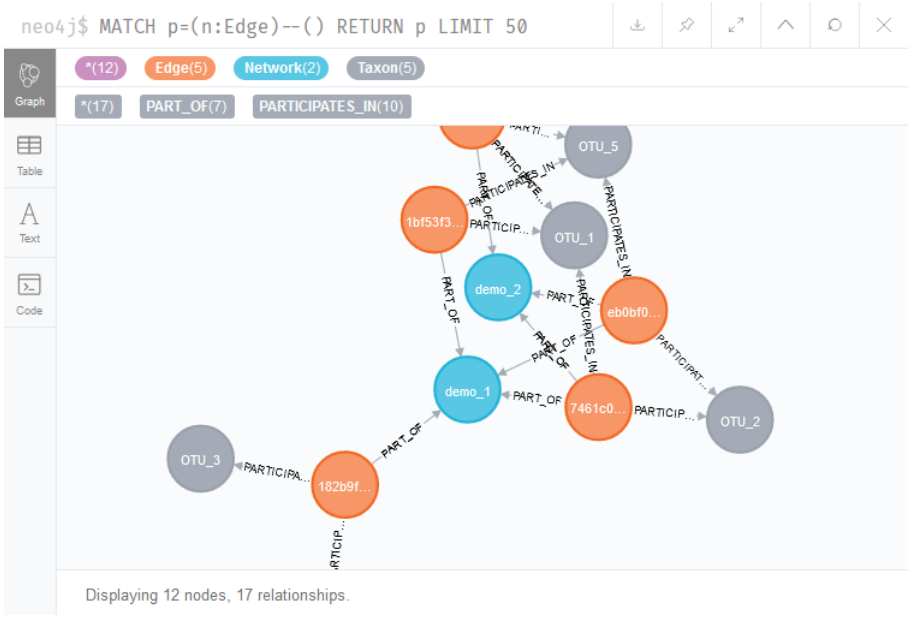

Figure 8: Edge links to taxa and networks.

4.4 Extracting the overlap between two networks

The command below generates a network containing all matching edges between **demo_1** and **demo_2**. By using the **-w** flag, mako is told to ignore whether edges have matching edge weights or not.

```
mako netstats -fp local\_filepath -cf
```

If you access the Neo4j Browser (http://localhost:7475/browser/) and run the following query, you should be able to access all the edges part of sets:

```
MATCH p=(n:Set)--() RETURN p LIMIT 50
```

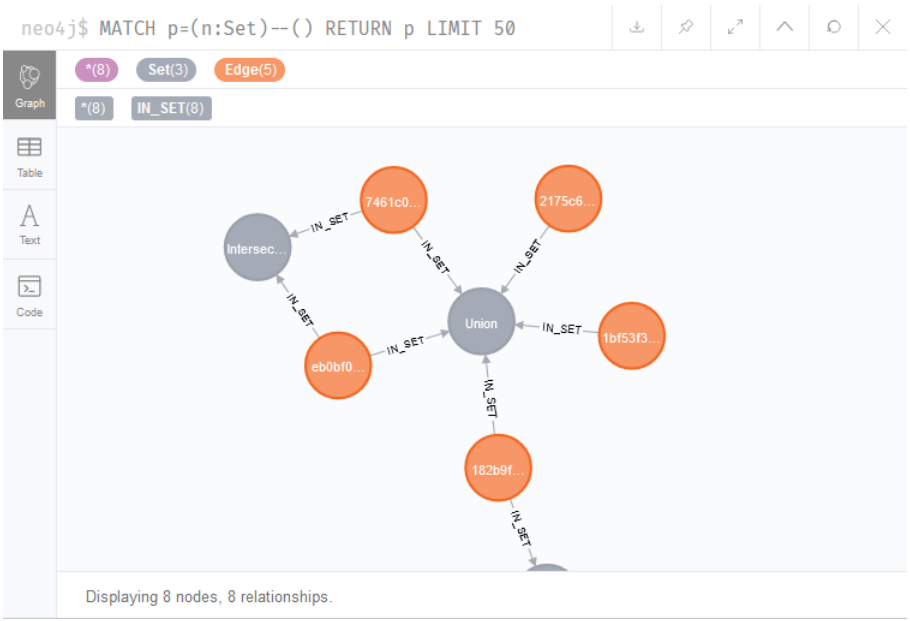

Figure 9: Network sets with edges.

4.5 Calculate metadata correlations

The command below carries out hypergeometric tests to see if the metadata (skin vs gut) is linked to specific taxa. Note that this is not a robust way to estimate correlations between taxa and metadata and dedicated, more appropriate statistical methods should be used for this purpose.

```
mako metastats -fp local\_filepath -cf -var all
```

If you access the Neo4j Browser (http://localhost:7475/browser/) and run the following query, you should be able to access all taxa that were linked to the skin or gut samples via the hypergeometric test:

```
MATCH p=(n:Taxon)--(:Property) RETURN p LIMIT 50
```

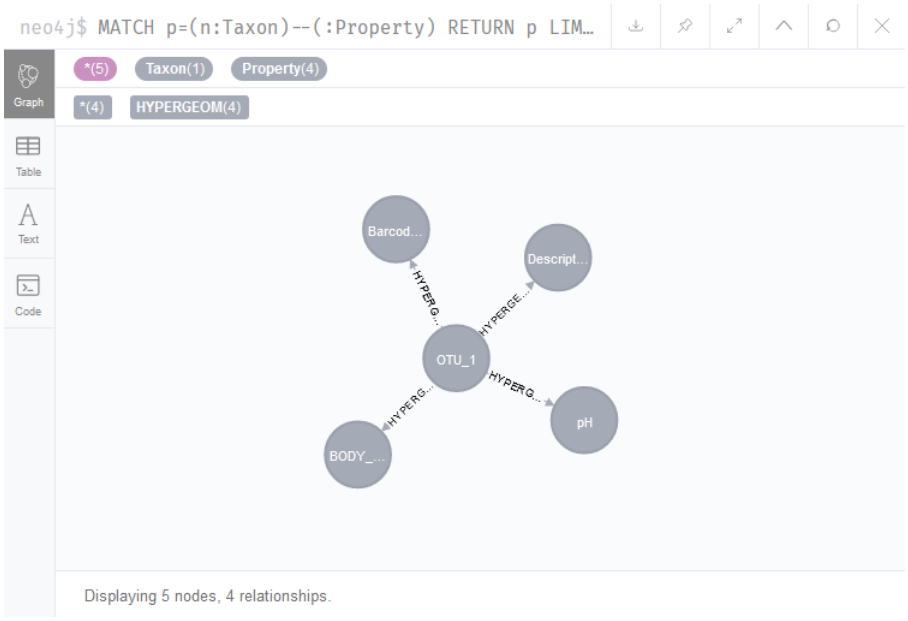

Figure 10: Metadata links to taxa.

4.6 Merge network to a higher taxonomic level

The command below merges networks by taxonomic level. For example, by merging to genus, both nodes assigned to *Escherichia* are merged into a single node, and by merging to order, the Halaerobiales nodes are merged as well. All lower taxonomic levels are also merged by running this command.

```
mako metastats -fp local\_filepath -cf -agglom order
```

If you access the Neo4j Browser (http://localhost:7475/browser/) and run the following query, you should be able to access one of the agglommerated networks:

```
MATCH p=(n:Network {name: 'Genus_demo_1'})--(:Property)
RETURN p LIMIT 50
```

4.7 Export a network to Cytoscape

Instead of working with the Neo4j database directly, it is also possible to export a specific network to a graphml file or to a running Cytoscape instance with

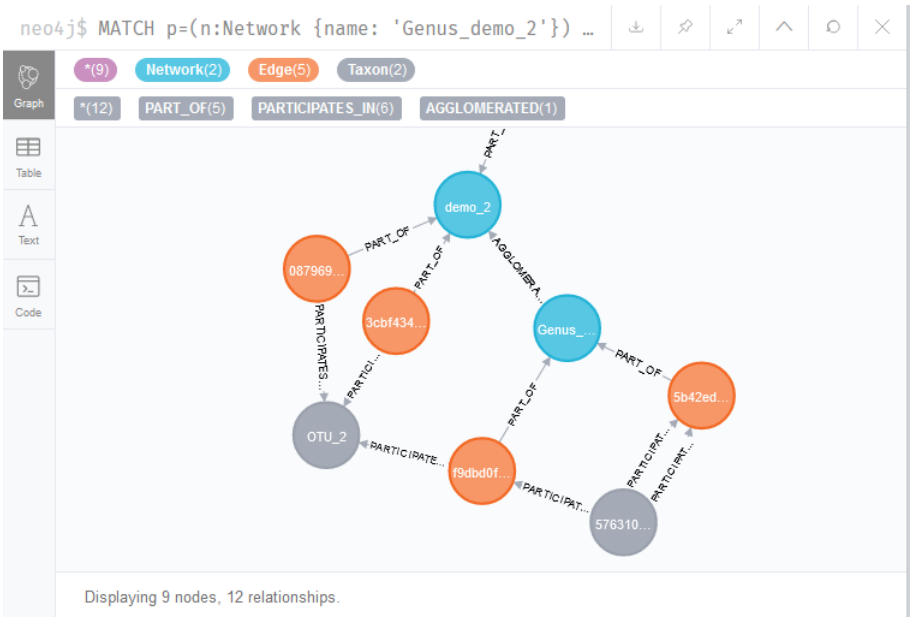

Figure 11: Network merged by taxonomy.

the commands below. Specify the target network with the **-net** flag. The new network is exported to the specified file path in **-fp**.

```
mako io -fp local\_filepath -cf -net Genus_demo_1 -cyto -write
```

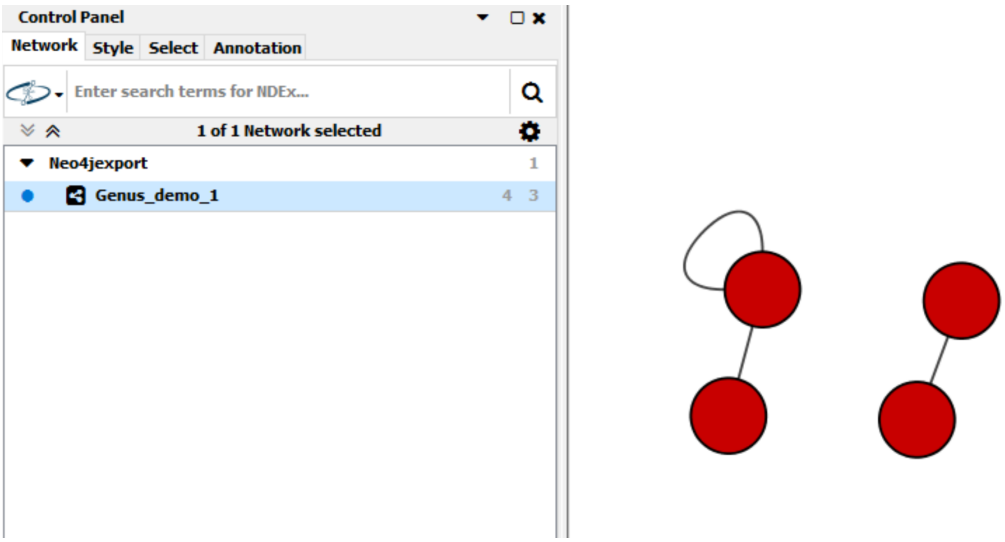

Figure 12: Cytoscape with a network imported from Neo4j.

5 mako - demo file with 60 networks

We can load a dump of mako's 60-network database into a Docker container. The following commands carry out such a task.

First, download the mako.dump file from the mako releases page. This file was created by uploading 60 BIOM files and network files to the Neo4j database and then using neo4j-admin to create a dump file. Such a file is much faster to restore a complete database.

First, a Docker container needs to be initialized and bound to a local volume. In the script below, the command has been given as a relative path (./data) to the working directory. Therefore, the mako.dump file needs to be saved in a folder called data and the Docker command needs to be called from the file location of the data folder.

```
docker run -d -v ./data:/data --publish=7475:7474
--publish=7688:7687 --name=neo4j --env NEO4J_AUTH=neo4j/test neo4j:3.5

docker stop neo4j
```

Next, the mako.dump file is copied from the data folder to Neo4j container.

```
docker cp ./data/mako.dump neo4j:/data/mako.dump
```

Then we restart the Neo4j docker, but append the script so that a bash shell is opened.

```
docker run -i -v ./data:/data --publish=7475:7474
--publish=7688:7687 -t neo4j /bin/bash
```

From the bash shell, we call the neo4j-admin script and use it to restore the neo4j database from the mako.dump file.

```
bin/neo4j-admin load --from=/data/mako.dump --database=neo4j --force
exit
```

Finally, we can restart the Docker container. Give it a minute or two to restore the database; afterwards, you should be able to interact with it via the Neo4j Browser at http://localhost:7475/browser/.

```
docker start neo4j
```

6 mako - manual

6.1 Introduction

The database schema used by mako defines a set structure for uploading microbiome-related files to a Neo4j database. The software tool is based around this data schema and provides an application programming interface (API), command line interface (CLI) and graphical user interface (GUI) for converting files to the database, or for extracting relevant files and exporting these as a text file or to Cytoscape. In this guide, we will briefly discuss mako's CLI, GUI and API.

6.2 CLI

The CLI has been separated in several modules for ease of use. Selecting a module is done by including the module name in the script call, like so:

- mako base
- mako neo4biom
- mako io
- mako netstats
- mako metastats
- mako manta
- mako anuran

For learning more about the parameters used by each module, type **mako *module* -h**. In the next sections, a brief overview of module functionalities is provided.

Several parameters are present across all modules. These provide information necessary to access the Neo4j database. By setting the *-cf* flag, Neo4j access information only needs to be stored once as mako will generate a config file that can be accessed. This is **not** a protected file, so this should not be used when addressing highly secured databases. However, for local Neo4j Desktop and Docker instances, the config file saves repeated entry of details.

- **-fp** File path for reading / writing files and log.
- **-cf** Use flag to store Neo4j access information.
- **-u** Neo4j username.
- **-p** Neo4j password.
- **-a** Neo4j address.

6.2.1 mako base

The **base** module can start a Neo4j Desktop if no Neo4j Desktop process has been started already, or safely shut down Neo4j Desktop if the process has been stored and the process ID is available in the config file. Additionally, the module can clear the database or check constraints.
If connecting to a Docker instance or other running instance of Neo4j, only the **-clear** and **-check** functions of the **base** module are relevant.

- **-n** Filepath to neo4j folder (only relevant for running Neo4j desktop).
- **-start** Start Neo4j Desktop. Flag **-cf** to store process ID.
- **-clear** Clears running Neo4j database.
- **-quit** Uses stored process ID in config to safely shut down Neo4j desktop.
- **-check** Checks if relationships in the database violate database schema constraints.

6.2.2 mako neo4biom

The **neo4biom** module can be used to write BIOM files or tab-delimited files to a Neo4j database according to mako's database schema. The **-fp** prefix contains the shared file path, so there is no need to write the full file path, partial file paths or just filenames are sufficient if **-fp** is used. For each file, an Experiment node is created that matches the filename. If more than one tab-delimited file is provided, metadata files should be in the same order as the count table. Instead of providing a count table with metadata, it also possible to create Taxon nodes from taxonomy tables alone. Additionally, the **neo4biom** module provides a function for deleting a single Experiment node; disconnected Taxon nodes no longer present in specimens are also removed.

- **-biom** One or more (filepaths of) BIOM files.
- **-count** One or more tab-delimited count tables (samples as columns).
- **-tax** One or more tab-delimited taxonomy tables.
- **-tm** One or more tab-delimited files with taxon metadata.
- **-sm** One or more tab-delimited files with sample metadata.
- **-del** Name(s) of Experiment node(s) that should be deleted.

6.2.3 mako io

The **io** module can be used to write network files to a Neo4j database according to mako's database schema. The **-fp** prefix contains the shared file path, so there is no need to write the full file path, partial file paths or just filenames are sufficient if **-fp** is used.

The **io** module also has options for adding extra metadata to the Neo4j database. If a tab-delimited edge list is supplied with existing nodes in the 1st column and new properties in the 2nd column, this is used to create a new Property node with relationships containing property values.

Finally, the module can delete and export networks or derived sets (generated with the **netstats** module) to a running instance of Cytoscape or to a graphml file. To run these operations, the **-net** parameter should be given names of Network or Set names in the Neo4j database.

- **-net** One or more (filepaths of) network files, or Network / Set names.
- **-del** Name(s) of Network or Set node(s) that should be deleted.
- **-fasta** Name(s) of FASTA files, e.g. GreenGenes FASTA files.
- **-cyto** If flagged, exports networks listed in **-net** to Cytoscape.
- **-meta** One or more tab-delimited files with node metadata (as a 2-column table). Column names are used to match node labels.
- **-w** If flagged, writes networks listed in **-net** to graphml.

6.2.4 mako netstats

The **netstats** module can be used to carry out set operations on networks in the Neo4j database, leading to creation of Set nodes that contain the intersections, unions or differences across (groups of) networks. If no networks are specified, operations are carried out across all networks.

By default, intersections only include edges with identical weights; so for an edge to be detected as present in 3 networks, it needs to have the same weight across all 3 networks. For the difference, if edges have different weights, they are considered unique edges so two edges with the same partners may be part of the difference. If the **-w** flag is used, this behaviour is reversed.

The fractions specified after **-frac** define partial intersections, e.g. all edges present in half of all networks.

- **-net** One or more Network names. The default takes all networks.
- **-w** If flagged, edge weights are not taken into account.
- **-frac** One or more fractions that are used to define subgroups for intersections.

6.2.5 mako metastats

The **metastats** module can carry out some basic operations on metadata. It can agglomerate networks to higher taxonomic levels (so all edges are merged to Family-level associations, for example), or it can carry out basic statistics. Specifically, it can calculate Spearman correlations between taxon abundances and quantitative metadata, and hypergeometric tests for qualitative metadata.

The edge agglomeration works by finding patterns, e.g. genus–taxon–edge–taxon–genus. If two edges have the same genus at both sides and matching weight, they can be merged together.

- **-net** One or more Network names to be used by -agglom. The default takes all networks.
- **-w** If flagged, edge weights are not taken into account.
- **-agglom** Choose a taxonomic level (species, genus, family, order, class, phylum) to agglomerate networks to.
- **-var** Specify "all" to carry out tests for all Property nodes connected to Specimen nodes, or one or more variable names to only test those variables.

6.2.6 mako anuran and manta

The **anuran** and **manta** modules can run anuran and manta on networks stored in the Neo4j database.

- **-net** One or more Network names to be supplied to **anuran** or **manta**
- **other flags** For all other flags, please look up the documentation for **anuran** and **manta**. Many of the default flags have been set to work well with most data.

6.3 GUI

The GUI provides a graphical interface for most of the functionality of the CLI.

6.4 API

While mako's functions can be called from command line, they have also been documented extensively so they can be used from a Python script. The **base**, **neo4biom**, **io**, **netstats**, **metastats** and **utils** modules each contain a class that can interact with a running Neo4j database instance. To carry out the functions as part of a script, simply instantiate an object with the necessary information to connect to the database and it will set up a Neo4j driver that can be used to run mako-specific functions or to run custom queries.

For example, the **io** module contains the **IoDriver** class. The IoDriver, like other **mako** drivers, can be called by providing Neo4j access details:

```
driver = IoDriver(user="testuser",
                  password="test",
                  uri=inputs['address'], filepath="C:/testfolder/",
                  encrypted=False)
driver.convert_networkx(network=networkx_object, network_id="test_network"
```

To see which class methods are available, import the class and then run **help(classname)**. The modules contain the following Neo4j-related classes:

- base - BaseDriver
- neo4biom - Biom2Neo
- io - IoDriver
- netstats - NetstatsDriver
- metastats - MetastatsDriver
- utils - ParentDriver

All Neo4j drives classes inherit methods from the ParentDriver class, and therefore have a query method that can be used to run Cypher queries. Queries can be passed as a string. If appliccable, dictionaries for parameterized batch queries can also be used.

References

[1] Natalya F Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W Fergerson, and Mark A Musen. Creating semantic web contents with protege-2000. *IEEE intelligent systems*, 16(2):60–71, 2001.

[2] Robert T Paine. Food web complexity and species diversity. *The American Naturalist*, 100(910):65–75, 1966.

[3] Nicholas Sioutos, Sherri de Coronado, Margaret W Haber, Frank W Hartel, Wen-Ling Shaiu, and Lawrence W Wright. Nci thesaurus: a semantic model integrating cancer-related clinical and molecular information. *Journal of biomedical informatics*, 40(1):30–43, 2007.