tethne Documentation

Release 0.3.1-alpha

Author

February 28, 2014

CONTENTS

Tethne provides tools for easily parsing and analyzing bibliographic data in Python. The primary emphasis is on working with data from the ISI Web of Science database, and providing efficient methods for modeling and analyzing citation-based networks. Future versions will include support for PubMed, Scopus, and other databases.

As of v0.3, Tethne is beginning to include methods for incorporating data from the JSTOR Data-for-Research service, and MALLET topic modeling.

Tethne relies on NetworkX for graph classes, and leverages its network analysis algorithms. You can visualize networks produced with Tethne in Cytoscape or Gephi.

To get started, consult the tutorial. For support, visit our GitHub repository.

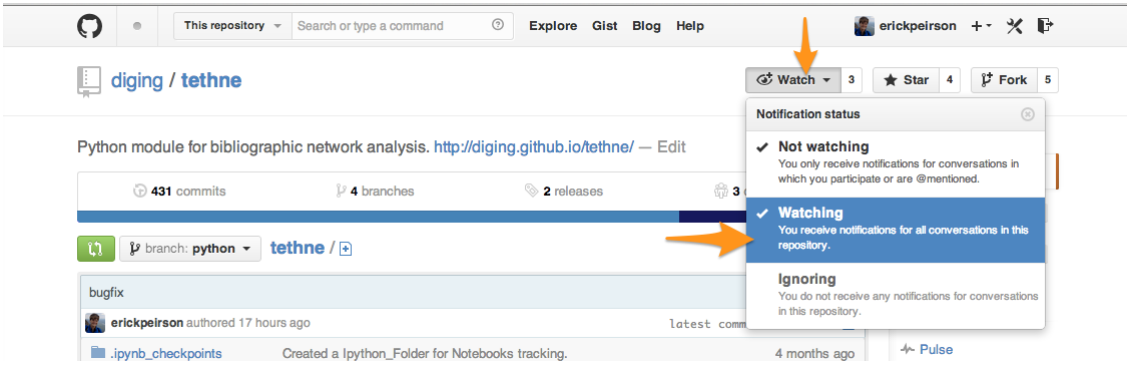
CHAPTER

ONE

CONTENTS

1.1 Installation

The following sections describe how to install the Tethne package and TethneGUI. Since Tethne is under active development, we're making improvements and adding features all the time. It's a good idea to stay on top of new releases. To get notifications about new releases, you should watch our GitHub repository.



1. Find our GitHub repository at https://github.com/diging/tethne
2. Click the **Watch** button in the upper-right corner of the page, and select **Watching**.

1.1.1 How to get help

Tethne is under active development, so you will most certainly run into bugs and hiccups. If you run into trouble, please create a new issue in our issue tracker. This will help us to hunt down problems, and will ensure that you get updates as we work to solve those problems.

1.1.2 Requirements

Tethne requires the following software and packages.:

- *Python 2.7*
- *NumPy*
- *NLTK*
- *NetworkX*

Anaconda

We recommend installing Anaconda, which will install Python, Numpy, NLTK, and a variety of other useful libraries. Installation instructions for Anaconda can be found here. This may be a good idea *even if you already have Python installed on your system*.

Python 2.7

Tethne requires Python 2.7; Python 3 is not fully backwards-compatible, and Tethne will not work properly with that version. **If you installed** Anaconda, **Python 2.7 should already be installed.**

You may already have Python installed on your system. To find out, open a new command-line window. On Mac, find `Terminal` in `Applications > Utilities`; on Windows, go to `Start > All Programs > Accessories > Command Prompt`. Type in `python` and press enter. If Python is installed, the command-line interpreter should start:

```
$ python
Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Note that the version number (`Python 2.7.5`) is listed in the first line when the interpreter starts. If you have Python 2.7.x, you're good to go.

If Python is not installed, or you have the wrong version of Python, you can find the latest versions of Python here.

Windows users should use the 32-bit version of Python. Look for the `Windows x86 MSI Installer`. For details about installing Python on Windows, see Installing Python on Windows.

NumPy

NumPy stands for Numerical Python. NumPy is a Python library that provides functionality for scientific computing.

If you installed Anaconda, **NumPy should already be installed**. Otherwise, you can find an installer for your operating system on the NumPy SourceForge project. Mac users should download and install `numpy-1.8.0-py2.7-python.org-macosx10.6.dmg`. Windows users should download and install `numpy-1.8.0-win32-superpack-python2.7.exe`.

NLTK

NLTK stands for Natural Language ToolKit. **If you installed** Anaconda, **NLTK should already be installed.** For installation instructions, see the NLTK documentation.

NetworkX

NetworkX is a Python package for network analysis. The easiest way to install NetworkX is to use `pip`. `pip` installs Python packages from the Python Package Index. You will need to be connected to the internet in order for `pip` to successfully download and install packages.

First, check to see whether you have `pip` installed. Open the command prompt, and enter `pip --version`. If `pip` is installed, you should see something like:

```
$ pip --version
pip 1.3.1 from /anaconda/lib/python2.7/site-packages (python 2.7)
```

Otherwise, you'll need to install `pip`. See this installation guide.

Once `pip` is installed, install NetworkX by entering the following in the command prompt:

```
$ sudo pip install networkx
```

You will be prompted to enter your password.

If all goes well, you should be able to `import` NetworkX in Python:

```
$ python
 Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
 [GCC 4.0.1 (Apple Inc. build 5493)] on darwin
 Type "help", "copyright", "credits" or "license" for more information.
 >>> import networkx
 >>>
```

1.1.3 Tethne for Python and Command-line

From PyPI

Tethne is available via the Python Package Index. You can install the latest release using `pip`

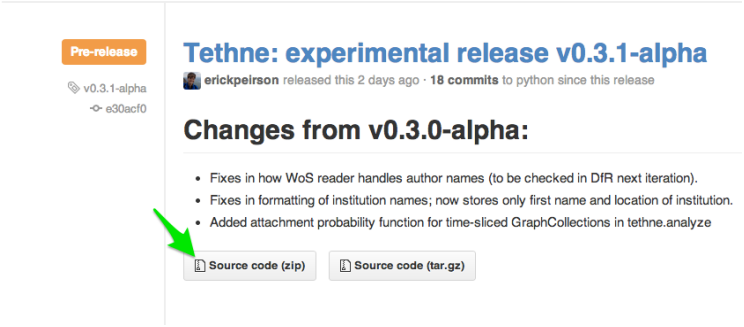
```
$ sudo pip install tethne
Password:
Downloading/unpacking tethne
  Downloading tethne-0.3.0-alpha.tar.gz (61kB): 61kB downloaded
  Running setup.py egg_info for package tethne

Installing collected packages: tethne
  Running setup.py install for tethne

Successfully installed tethne
Cleaning up...
```

From GitHub

Alternatively, you can find the latest release of Tethne in our GitHub repository.



1. Download the source code.
2. Unpack the .zip/.tar.gz archive (e.g. `tethne-0.3.1-alpha.zip`). This should create a new folder, e.g. `tethne-0.3.1-alpha`.
3. Open the command prompt, and navigate to the folder where you unpacked Tethne. For example, if you unpacked Tethne in your Downloads folder, use:

```
$ cd ~/Downloads
```

4. If you have `pip` installed, use:

```
$ sudo pip install ./tethne-0.3.1-alpha
```

(change `tethne-0.3.1-alpha` to reflect the release that you downloaded).

5. If you don't have `pip`, you can use:

```
$ sudo python ./tethne-0.3.1-alpha/setup.py install
```

If Tethne is installed successfully, you should be able to `import` it in the Python interpreter:

```
$ python
Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tethne.readers as rd
>>>
```

Alias

To make using Tethne from the command-line a bit easier, create a permanent alias.

Mac:

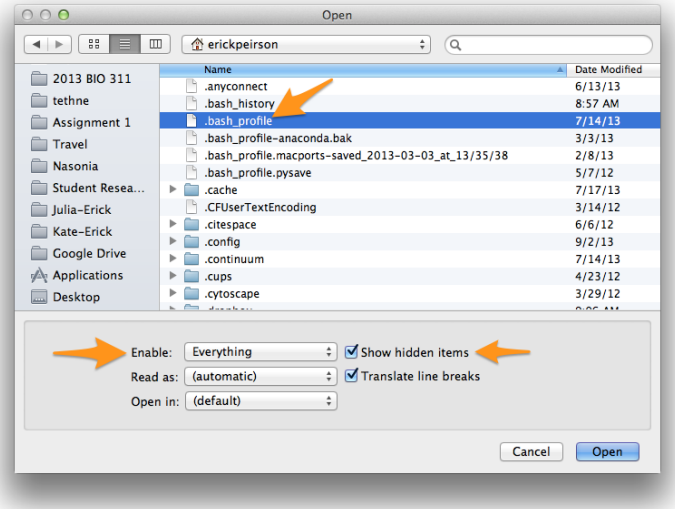
Add the following line to `~/.bash_profile`:

```
alias tethne='python [TETHNE PATH]'
```

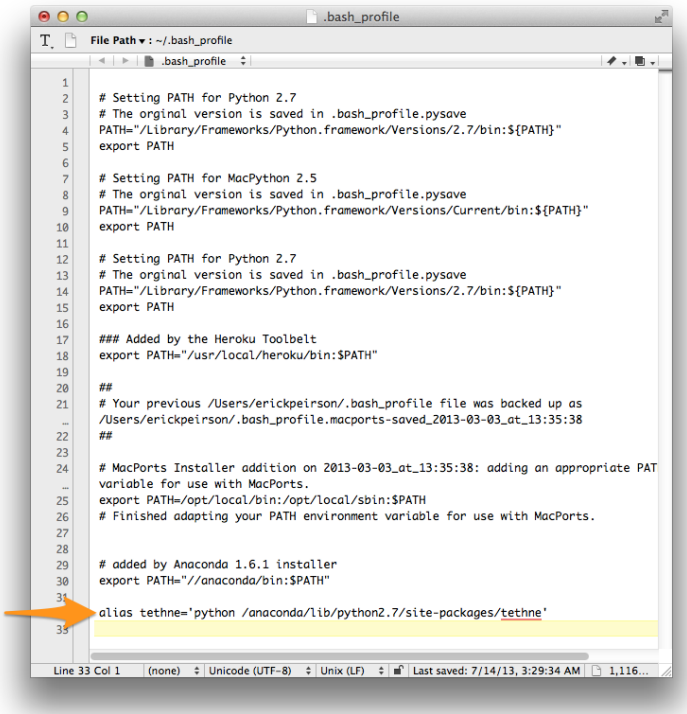
1. To find the `[TETHNE PATH]`, start the Python interpreter in Terminal, and import Tethne. Then call `tethne.__file__`. In the example below, the path that we're looking for is `/anaconda/lib/python2.7/site-packages/tethne`:

```
$ python
Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tethne
>>> tethne.__file__
'//anaconda/lib/python2.7/site-packages/tethne/__init__.pyc'
>>>
```

2. In TextWrangler, open the file `.bash_profile` in your Home directory. You may need to set `Enable` to `Everything`, and check the `Show hidden files` checkbox.



3. Add the line `alias tethne='python [TETHNE PATH]'` (change `[TETHNE PATH]` to the Tethne installation path, from step 1) to the end of `._bash_profile`, then save and close the file.



4. Open a new Terminal window, and enter `tethne`. If all goes well, you should see:

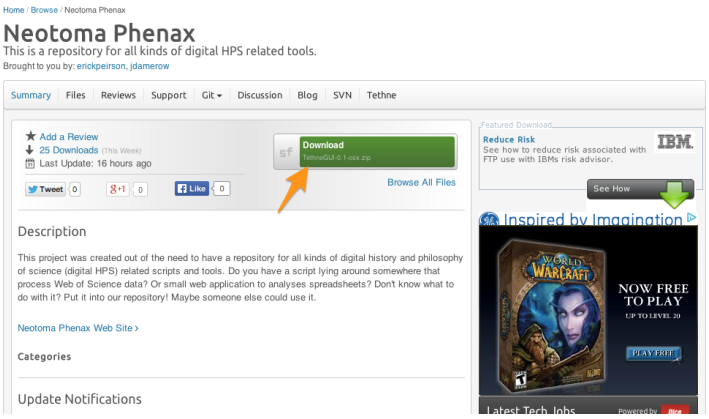
```
$ tethne
Must specify --dataset-id
```

1.1.4 TethneGUI

A demonstration graphical user interface was created for teaching purposes only. This is not intended for extensive use. It provides a very basic interface to the Tethne command-line workflow.

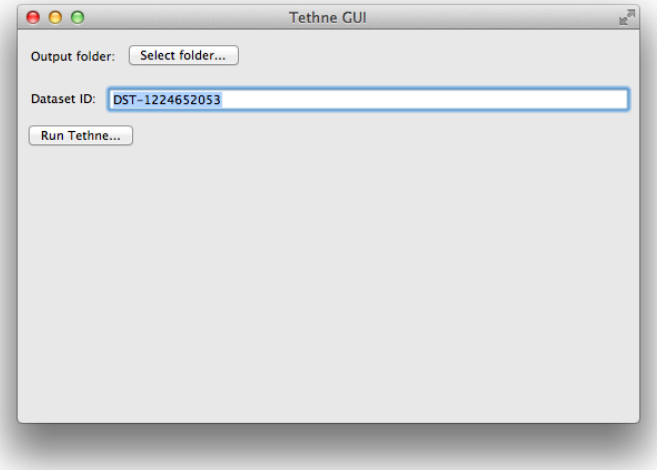
TethneGUI comes bundled with most of the things that it needs to run. On **Windows** systems, however, you will need to install NumPy independently. See *NumPy*, above.

You can find the most recent build of the TethneGUI on the Neotoma phenax SourceForge repository.



1. Download the version of TethneGUI appropriate for your operating system. Look for the green download button near the center of the page. Clicking this button will take you to a new page, and after a few seconds you should start downloading a .zip archive (e.g. `TethneGUI-0.1-osx.zip`)
2. Once the download completes, unpack it. This will create a new directory called `TethneGUI`. Move this directory to a place in your filesystem where you can find it later (e.g. your `Applications` folder).
3. Look inside the `TethneGUI` folder. **On Mac**, double-click `TethneGUI.app`, inside. **On Windows**, run `TethneGUI.exe`.

Once TethneGUI loads, you should see a window that looks like this:



1.2 Tutorial

Tethne is primarily developed as a Python package, but can also be invoked from the command-line (Mac OSX, Linux; untested for Windows).

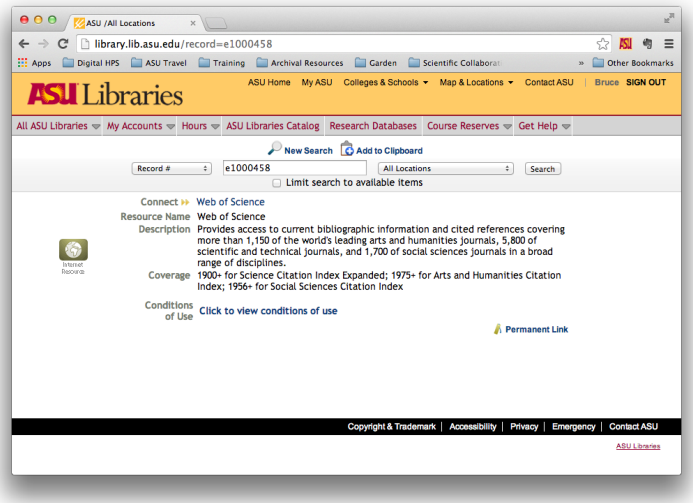
1.2.1 Getting Bibliographic Data

The current version of Tethne supports bibliographic data from the ISI Web of Science and JSTOR's Data-for-Research portal. Future releases will support data from PubMed and Scopus.

Web of Science

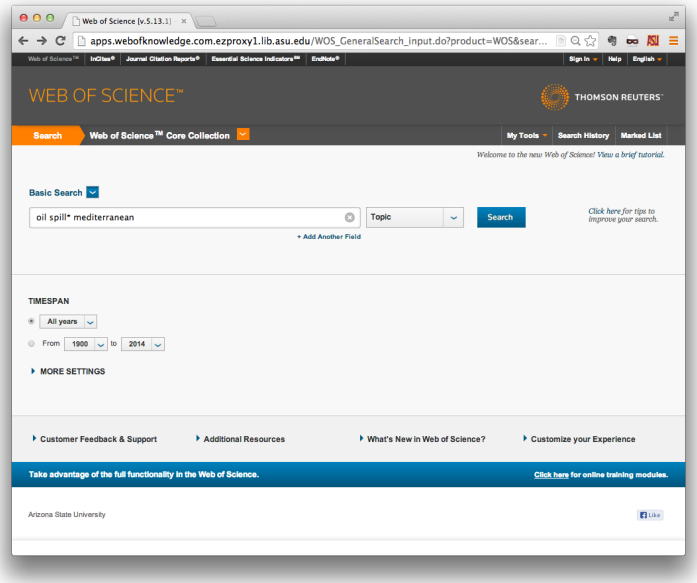
The ISI Web of Science is a proprietary database owned by Thompson Reuters. If you are affiliated with an academic institution, you may have access to this database via an institutional license.

To access the Web of Science database via the Arizona State University library, find the Web of Science entry in the library's online catalog. You may be prompted to log in to the University's Central Authentication System.



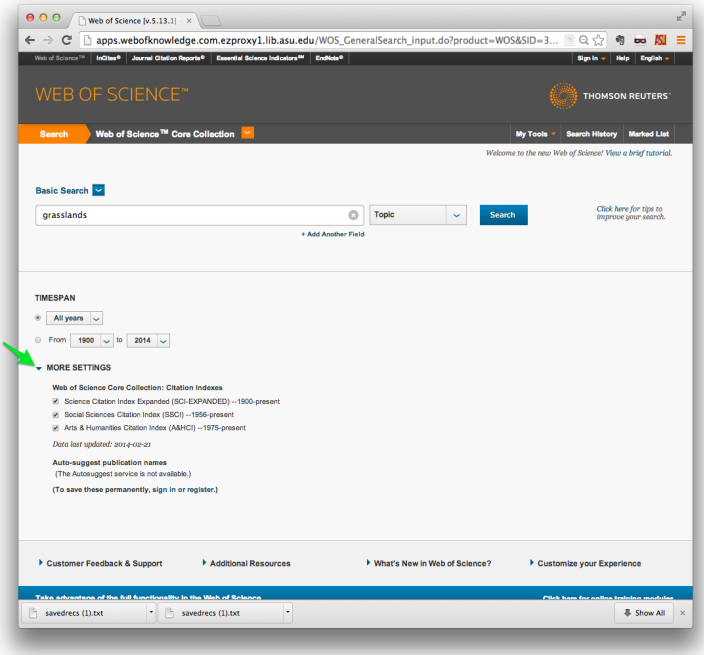
Perform a search for literature of interest using the interface provided.

Your search criteria will be informed by the objectives of your research project. If you are attempting to characterize the development of a research field, for example, you should choose terms that pick out that field as uniquely as possible (consider using the `Publication Name` search field). You can also pick out literatures originating from particular institutions, by using the `Organization-Enhanced` search field.

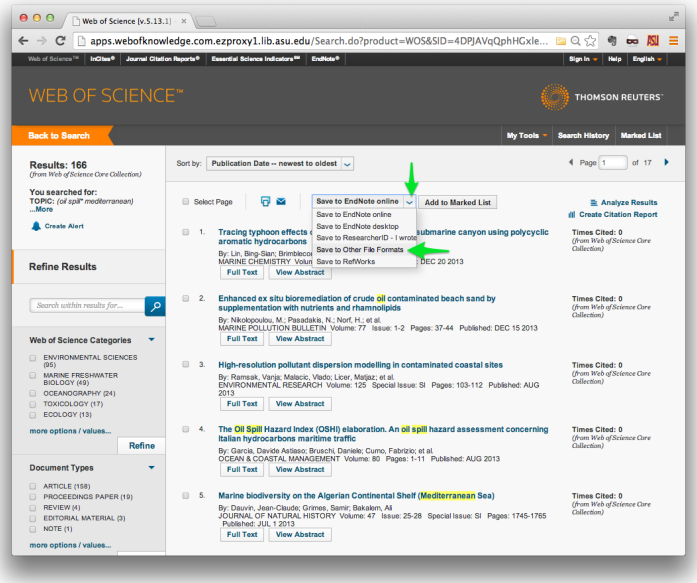


Note also that you can restrict your research to one of three indexes in the Web of Science Core Collection:

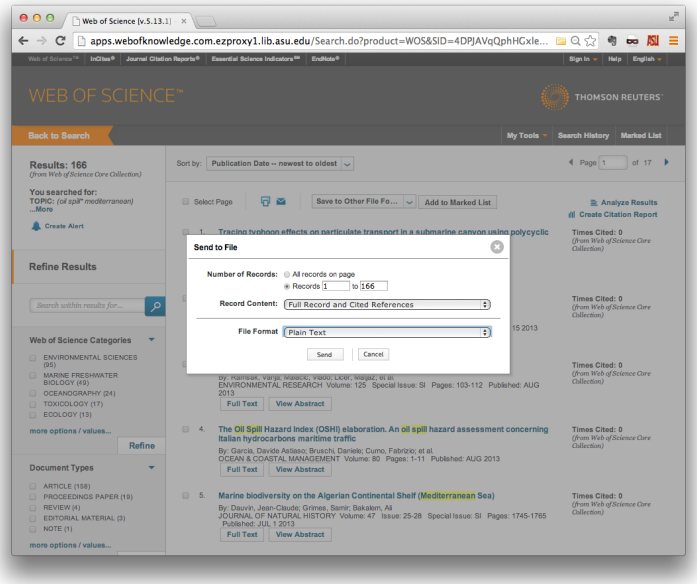
- Science Citation Index Expanded is the largest index, containing scientific publications from 1900 onward.
- Social Sciences Citation Index covers 1956 onward.
- Arts & Humanities Citation Index is the smallest index, containing publications from 1975 onward.



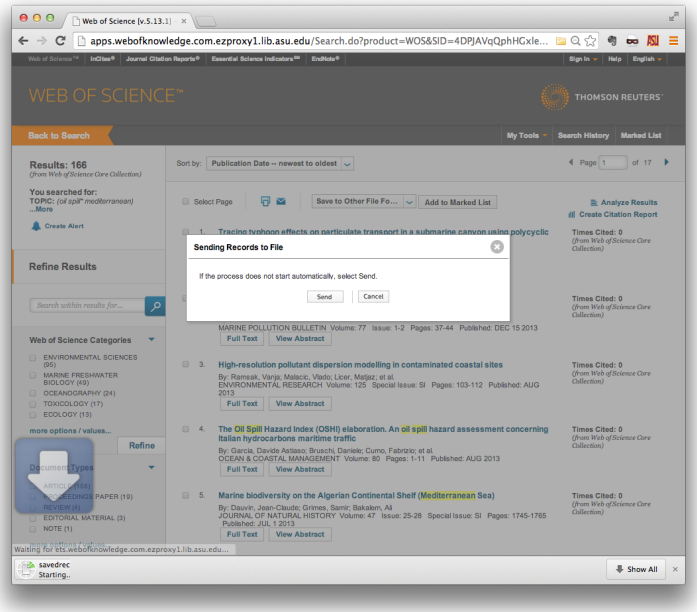
Once you have found the papers that you are interested in, find the `Send to:` menu at the top of the list of results. Click the small orange down-arrow, and select `Other File Formats`.



A small in-browser window should open in the foreground. Specify the range of records that you wish to download. **Note that you can only download 500 records at a time**, so you may have to make multiple download requests. Be sure to specify `Full Record and Cited References` in the *Record Content* field, and `Plain Text` in the *File Format* field. Then click `Send`.



After a few moments, a download should begin. WoS usually returns a field-tagged data file called `savedrecs.txt`. Put this in a location on your filesystem where you can find it later; this is the input for Tethne's WoS reader methods.



Structure of the WoS Field-Tagged Data File

If you open the text file returned by the WoS database (usually named 'savedrecs.txt'), you should see a whole bunch of field-tagged data. "Field-tagged" means that each metadata field is denoted by a "tag" (a two-letter code), followed by values for that field. A complete list of WoS field tags can be found here. For best results, you should avoid making changes to the contents of WoS data files.

The metadata record for each paper in your data file should begin with:

```
PT J
```

...and end with:

```
ER
```

There are two author fields: the AU field is always provided, and values take the form "Last, FI". AF is provided if author full-names are available, and values take the form "Last, First Middle". For example:

```
AU Dauvin, JC
   Grimes, S
   Bakalem, A
AF Dauvin, Jean-Claude
   Grimes, Samir
   Bakalem, Ali
```

Citations are listed in the CR block. For example:

```
CR Airoldi L, 2007, OCEANOGR MAR BIOL, V45, P345
   Alexander Vera, 2011, Marine Biodiversity, V41, P545, DOI 10.1007/s12526-011-0084-1
   Arvanitidis C, 2002, MAR ECOL PROG SER, V244, P139, DOI 10.3354/meps244139
   Bakalem A, 2009, ECOL INDIC, V9, P395, DOI 10.1016/j.ecolind.2008.05.008
   Bakalem Ali, 1995, Mesogee, V54, P49
   ...
   Zenetos A, 2005, MEDITERR MAR SCI, V6, P63
   Zenetos A, 2004, CIESM ATLAS EXOTIC S, V3
```

More recent records also include the institutional affiliations of authors in the C1 block.

```
C1 [Wang, Changlin; Washida, Haruhiko; Crofts, Andrew J.; Hamada, Shigeki;
Katsube-Tanaka, Tomoyuki; Kim, Dongwook; Choi, Sang-Bong; Modi, Mahendra; Singh,
Salvinder; Okita, Thomas W.] Washington State Univ, Inst Biol Chem, Pullman, WA 99164
USA.
```

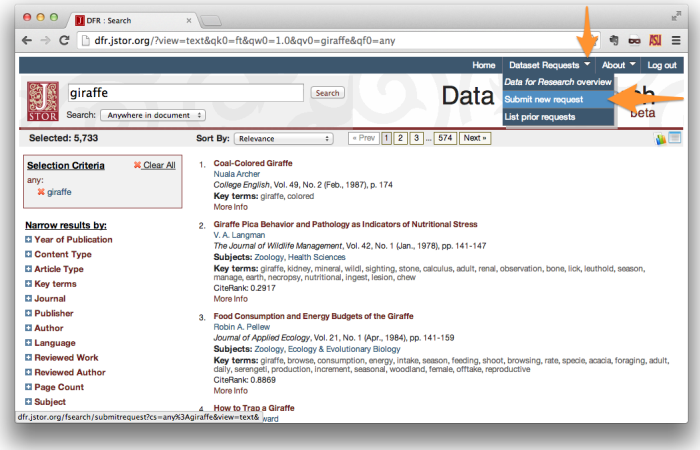
For more information about WoS field tags, see a list on the Thompson Reuters website, here.

JSTOR Data-for-Research

The JSTOR Data-for-Research (DfR) portal gives researchers access to bibliographic data and N-grams for the entire JSTOR database. Tethne can use DfR data to generate coauthorship networks, and to improve metadata for Web of Science records. Increasingly, Tethne is also able to use N-gram counts to add information to networks, and can generate corpora for common topic modeling tools (coming soon!).

Access the DfR portal at http://dfr.jstor.org/ If you don't already have an account, you will need to create a new account.

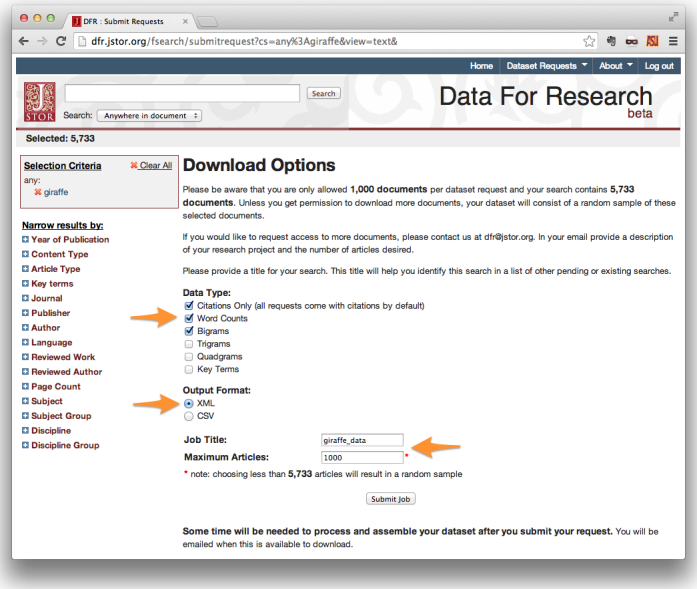
After you've logged in, perform a search using whatever criteria you please. When you have achieved the result that you desire, create a new dataset request. Under the "Dataset Request" menu in the upper-right corner of the page, click "Submit new request".



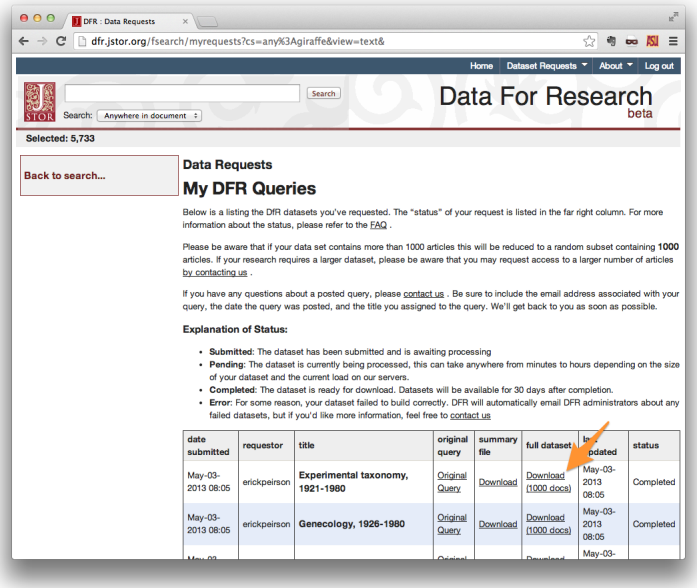
On the **Download Options** page, select your desired **Data Type**. If you do not intend to make use of the contents of the papers themselves, then "Citations Only" is sufficient. Otherwise, choose word counts, bigrams, etc.

Output Format should be set to **XML**.

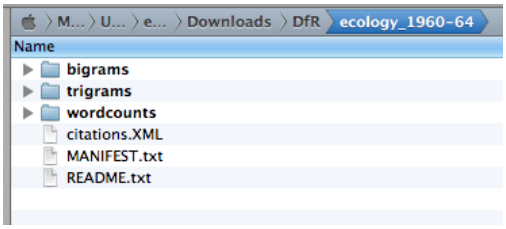
Give your request a title, and set the maximum number of articles. *Note that the maximum documents allowed per request is 1,000. Setting **Maximum Articles** to a value less than the number of search results will yield a random sample of your results.*



Your request should now appear in your list of **Data Requests**. When your request is ready (hours to days later), you will receive an e-mail with a download link. When downloading from the **Data Requests** list, be sure to use the link in the **full dataset** column.



When your dataset download is complete, unzip it. The contents should look something like those shown below.



`citations.XML` contains bibliographic data in XML format. The `bigrams`, `trigrams`, `wordcounts` folders

contain N-gram counts for each document.

In the example above, the path this dataset is */Users/erickpeirson/Downloads/DfR/ecology_1960-64*. This is the path used in `tethne.readers.dfr.read()` .

1.2.2 Quickstart (Command-line)

Use the following sequence of commands to generate a dynamic co-authorship network using data from the ISI Web of Science database. The examples below are from the Mac terminal, but should work on the Windows command-prompt as well.

For detailed documentation of command-line options, see *Command-line Options*.

Tethne is invoked using `python ./tethne`, where `./tethne` is the path to the Tethne

1. *Read*

Tethne can parse data from the Web of Science, JSTOR Data-for-Research, and a few other sources.

```
$ python ./tethne -I example_data -O ./ --read-file \
> -P /Users/erickpeirson/Desktop/savedrecs (101).txt -F WOS

----------------------------------------
        Workflow step: Read
----------------------------------------
Reading WOS data from file /Users/erickpeirson/Desktop/savedrecs.txt...done.
Read 500 papers in 1.42379593849 seconds. Accession: 90a0e7fe-c081-4749-9e7c-43534d9b9558.
Generating a new DataCollection...done.
Saving DataCollection to /tmp/example_data_DataCollection.pickle...done.
```

`-I example_data` tells Tethne to use the ID `example_data` for this dataset. This should be used for each workflow step. `-O ./` tells Tethne to save output (e.g. statistics and networks) to the current working directory.

`-F WOS` tells Tethne that the data is in Web of Science field-tagged format.

2. *Slice*

The `slice` step tells Tethne how to partition your dataset for analysis.

If you are studying network evolution over time, your first slice axis will almost always be `date`. In the example below, `-S date,jtitle` tells Tethne to slice first by `date`, then by `jtitle`.

```
$ python ./tethne -I example_data -O ./ --slice -S date,jtitle -M time_period \
> --slice-window-size=2 --cumulative

----------------------------------------
        Workflow step: Slice
----------------------------------------
Loading DataCollection from /tmp/example_data_DataCollection.pickle...done.
Slicing DataCollection by date...done.
Slicing DataCollection by jtitle...done.
Saving slice distribution to .//example_data_sliceDistribution.csv...done.
Saving sliced DataCollection to /tmp/example_data_DataCollection_sliced.pickle...done.
```

`-M time_period --slice-window-size=2` tells Tethne to divide the dataset up into two-year time-periods. `--cumulative` means that each time-period will include data from all of the earlier time-periods.

3. *Graph*

The `graph` step generates networks from your data (one network per slice).

```
$ python ./tethne -I example_data -O ./ --graph -N author -T coauthors \
> --edge-attr=date,jtitle,ayjid

----------------------------------------
        Workflow step: Graph
----------------------------------------
Loading DataCollection with slices from /tmp/example_data_DataCollection_sliced.pickle...done.
Using first slice in DataCollection: date.
Building author graph using coauthors method...done in 1.90734863281e-05 seconds.
Saving GraphCollection to /tmp/example_data_GraphCollection.pickle...done.
Writing graph summaries to .//example_data_graphs.csv...done.
```

`-N author -T coauthors` tells Tethne to generate a coauthorship network, where nodes are authors. `--edge-attr=date,jtitle` tells Tethne to add the publication date and journal to each coauthorship edge.

Adding `--merged` would tell Tethne to ignore slicing and create a single network from the whole dataset.

4. *Analyze*

The `analyze` step is optional. This uses methods from NetworkX and the `tethne.analyze` module to analyze your networks.

```
$ python ./tethne -I example_data -O ./ --analyze -A betweenness_centrality

----------------------------------------
        Workflow step: Analyze
----------------------------------------
Loading GraphCollection from /tmp/example_data_GraphCollection.pickle...done.
Analyzing GraphCollection with betweenness_centrality...done.
Writing graph analysis results to .//example_data_betweenness_centrality_analysis.csv...done.
Saving GraphCollection to /tmp/example_data_GraphCollection.pickle...done.
```

`-A betweenness_centrality` tells Tethne to calculate the betweenness centrality of each node in each network, and save those values as node attributes.

5. *Write*

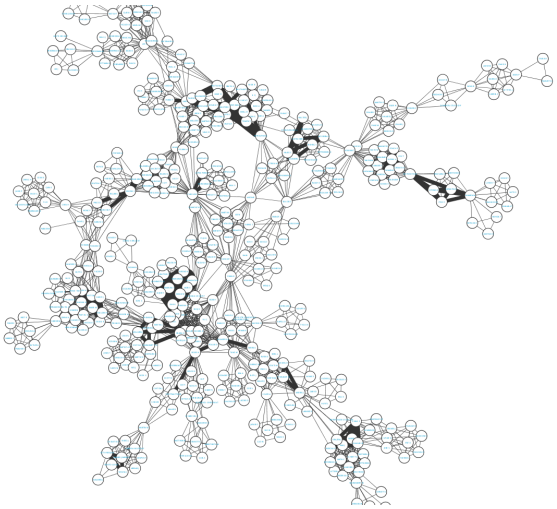
Tethne can write networks to a few different formats for visualization in Cytoscape or Gephi.

```
$ python ./tethne -I example_data -O ./ --write -W xgmml

----------------------------------------
        Workflow step: Write
----------------------------------------
Loading GraphCollection from /tmp/example_data_GraphCollection.pickle...done.
Writing graphs to ./ with format xgmml...done.
```

`-W xgmml` tells Tethne to generate a dynamic network in XGMML format.

The resulting graph might look something like (edge width <- N coauthored papers):



For detailed descriptions of each workflow step, see **'Step-By-Step Guide (Command-line)'_**.

1.2.3 Quickstart (Python)

After starting Python, import Tethne modules with:

```
>>> import tethne.readers as rd
```

To parse some data from the Web of Science, use the `tethne.readers.wos` module. See the `Paper` class for more information about what Tethne extracts from your WoS data.

```
>>> papers = rd.wos.read("/Path/To/FirstDataSet.txt")
>>> papers[0]
<tethne.data.Paper instance at 0x1015ed200>
```

You can either generate a network directly from your data using the methods in `tethne.networks.wos` , or you can package your data in a `DataCollection` for comparative or longitudinal analysis. If you want to generate a dynamic network, then use a `DataCollection` .

Simple Networks

To generate a single network directly from your data, use the `tethne.networks.wos` module. For example, to build a bibliographic coupling network:

```
>>> import tethne.networks as nt
>>> BC = nt.citations.bibliographic_coupling(papers)
>>> BC
<networkx.classes.graph.Graph object at 0x101b4fe50>
```

NetworkX provides algorithms for graph analysis. To generate betweenness-centrality values for each node, try:

```
>>> import networkx as nx
>>> b_centrality = nx.betweenness_centrality(BC)
>>> nx.set_node_attributes(BC, 'betweenness', b_centrality)
>>> BC.nodes(data=True)[0]
('BRADSHAW 1965 ADV GENET', {'betweenness': 0.12345})
```

You can then export your network for visualization using one of the methods in `tethne.writers` . For example, to generate a simple interaction file (SIF):

```
>>> import tethne.writers as wr
>>> wr.graph.to_sif(BC, '/Path/to/Output/File')
```

Sliced Networks for Comparative or Longitudinal Analysis

Especially if you want to create a dynamic network (a network that changes over time), use a `DataCollection` to organize your data.

To create a `DataCollection`:

```
>>> from tethne.data import DataCollection, GraphCollection
>>> D = DataCollection(papers)
```

You can then use the `tethne.data.DataCollection.slice()` method to slice your data, e.g. by publication date. To use a 4-year sliding time-window:

```
>>> D.slice('date', 'time_window', window_size=4)
```

Create a `GraphCollection` to manage your graphs. `tethne.builders` provides some helpful classes for creating a `GraphCollection` from a `DataCollection`. For example `authorCollectionBuilder` build a `GraphCollection` using an author-based network (e.g. coauthorship networks) in `tethne.networks.authors`:

```
>>> from tethne.builders import authorCollectionBuilder
>>> builder = authorCollectionBuilder(D)
>>> C = builder.build('date', 'coauthors')
```

To write dynamic XGMML for visualization in Cytoscape, use the writing methods in `tethne.writers.collection`:

```
>>> import tethne.writers as wr
>>> wr.collection.to_dxgmml(C, '/Path/to/Network.xgmml')
```

1.2.4 Coauthorship Networks

*This tutorial was developed for the course Introduction to Digital & Computational Methods in the Humanities (HPS), created and taught by Julia Damerow and Erick Peirson.

Coauthorship networks are among the most popular models for studying the structure of research communities, due in no small part to the ease with which coauthorship networks can be generated.

In this tutorial, we will generate a coauthorship network using data from the ISI Web of Science database. See *Getting Bibliographic Data*.

Each step includes instructions for the Tethne command-line, the TethneGUI, and Python. Command-line steps assume that you have created an *Alias* for Tethne.

If you run into problems, don't panic. Tethne is under active development, and there are certainly bugs to be found. Please report any problems on our GitHub issue tracker.

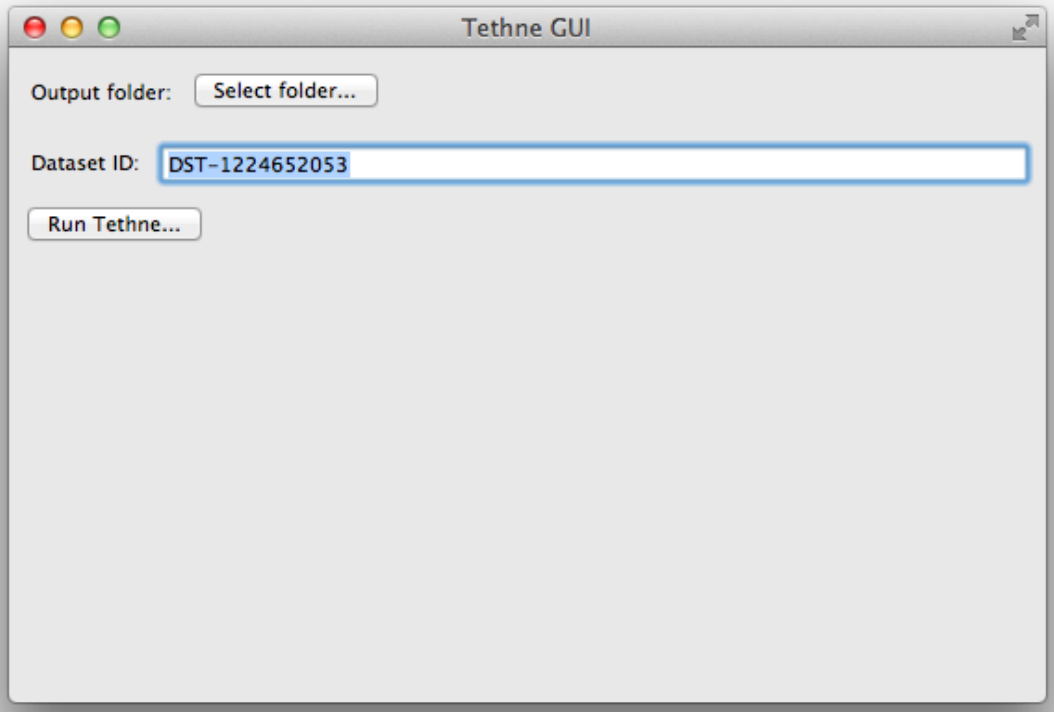
Getting Started

Before you start, you should choose an output folder where TethneGUI should store graphs and descriptions of your dataset.

You should also choose a dataset ID. This is a unique ID that Tethne will use to keep track of your data between workflow steps.

Initialize TethneGUI

When you first start TethneGUI, you should see a window like the one shown below. Click `Select folder...` to specify your output folder. A dataset ID should be automatically generated for you; you can change this if you wish.



Once you've selected an output folder and a dataset ID, click the `Run Tethne...` button.

Reading WoS Data

You can read WoS data from one or multiple field-tagged data files.

Command-line

Use `-I examplID` to specify your dataset ID, and `-O /Users/erickpeirson/exampleOutput` to specify your output folder.

`--data-format=WOS` tells Tethne that your data are in the Web of Science field-tagged format.

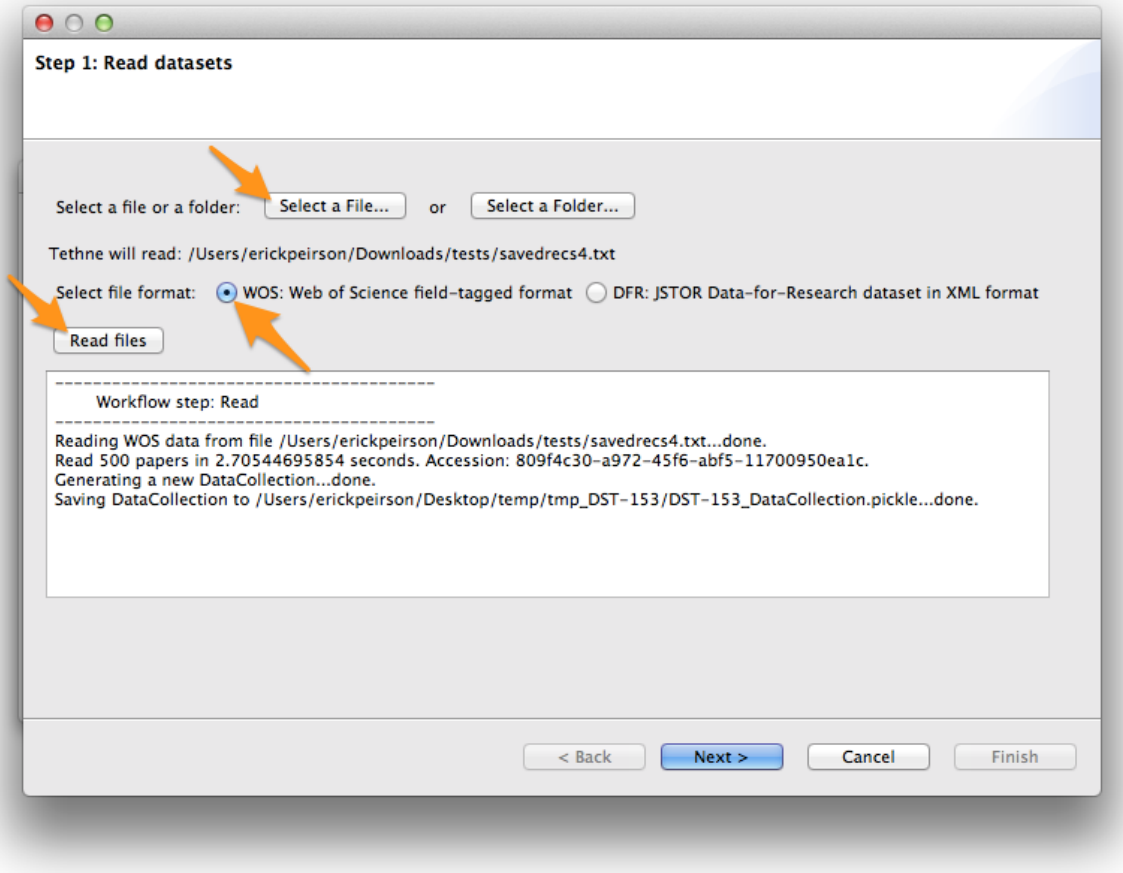
```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --read-file \
--data-path=/Users/erickpeirson/Downloads/tests/savedrecs4.txt --data-format=WOS
----------------------------------------
        Workflow step: Read
----------------------------------------
Reading WOS data from file /Users/erickpeirson/Downloads/tests/savedrecs4.txt...done.
```

```
Read 500 papers in 2.67462515831 seconds. Accession: 0ff65dc3-b8f7-4bdc-a714-2d2a539f10a9.
Generating a new DataCollection...done.
Saving DataCollection to /tmp/exampleID_DataCollection.pickle...done.
```

TethneGUI

1. Select your WoS data file. If you have one data file, click the `Select a File...`. If you have multiple data files in their own folder, click `Select a Folder...`.
2. Select the `WOS` file format.
3. Click the `Read files` button.

Depending on the size of your dataset, this may take a minute or two. When TethneGUI is done reading your data, you should see messages like those depicted in the image below.



If your data are read successfully, click `Next >`.

Python

First import the `tethne.readers` module, then use the `readers.wos.read()` method to create a list of `Paper` instances. You can use `readers.wos.from_dir()` to import all of the WoS datafiles in a directory.

```
>>> # Parse data.
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/To/FirstDataSet.txt")
```

Then create a new `DataCollection` to organize your data.

```
>>> from tethne.data import DataCollection
>>> D = DataCollection(papers)
```

Slicing WoS Data

In this tutorial, we will analyze the evolution of a coauthorship network over time. To do this, we will slice our data using the `date` field of each paper in our dataset.

We'll use the `time_period` slice method, which means that the data will be divided into subsets each containing data from a particular time period. The default window size is 1, and the window will advance by 1 year in each slice.

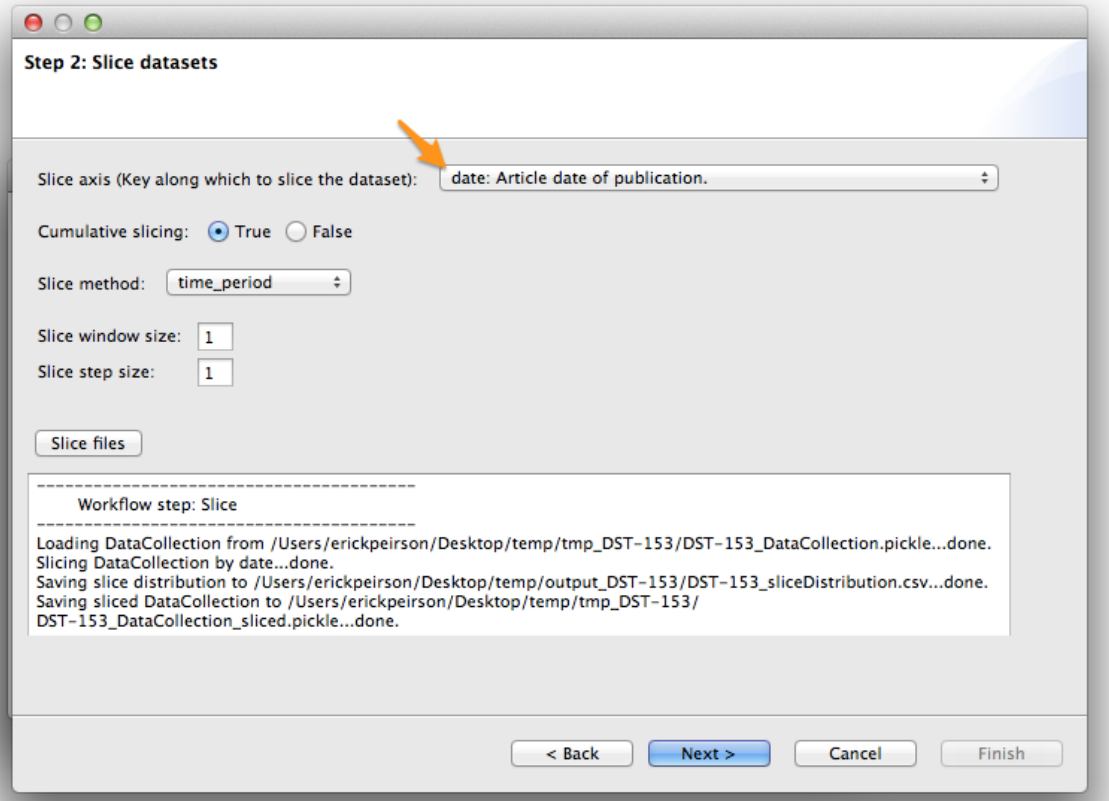
We'll also use the cumulative slicing option, which means that the data from each time period will contain data from all of the previous time-periods. In other words, the 1957 subset will contain data from 1957, and the 1958 subset will contain data from 1957 and 1958.

Command-line

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --slice -S date \
> -M time_period --cumulative
----------------------------------------
        Workflow step: Slice
----------------------------------------
Loading DataCollection from /tmp/exampleID_DataCollection.pickle...done.
Slicing DataCollection by date...done.
Saving slice distribution to /Users/erickpeirson/exampleOutput/exampleID_sliceDistribution.csv...done
Saving sliced DataCollection to /tmp/exampleID_DataCollection_sliced.pickle...done.
```

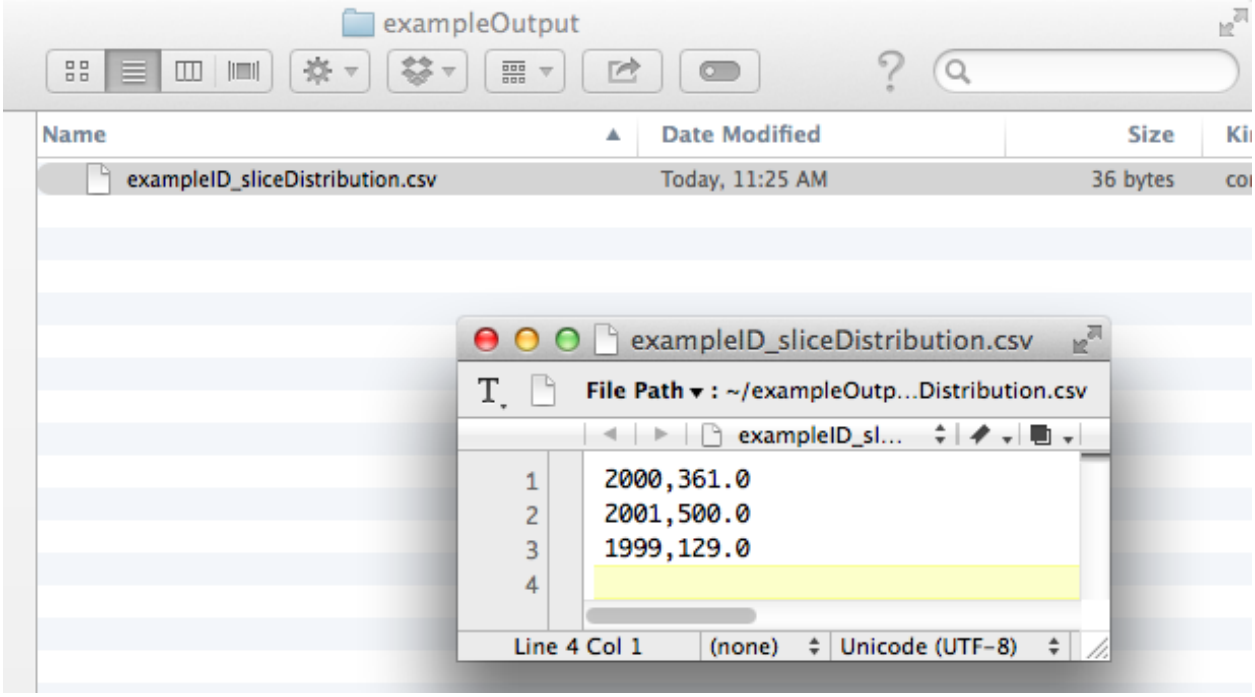
TethneGUI

The slice axis should be set to `date` by default. If not, select it from the `Slice axis` drop-down menu. Then click the `Slice files` button. After a few minutes, slicing should be complete; click `Next >`.

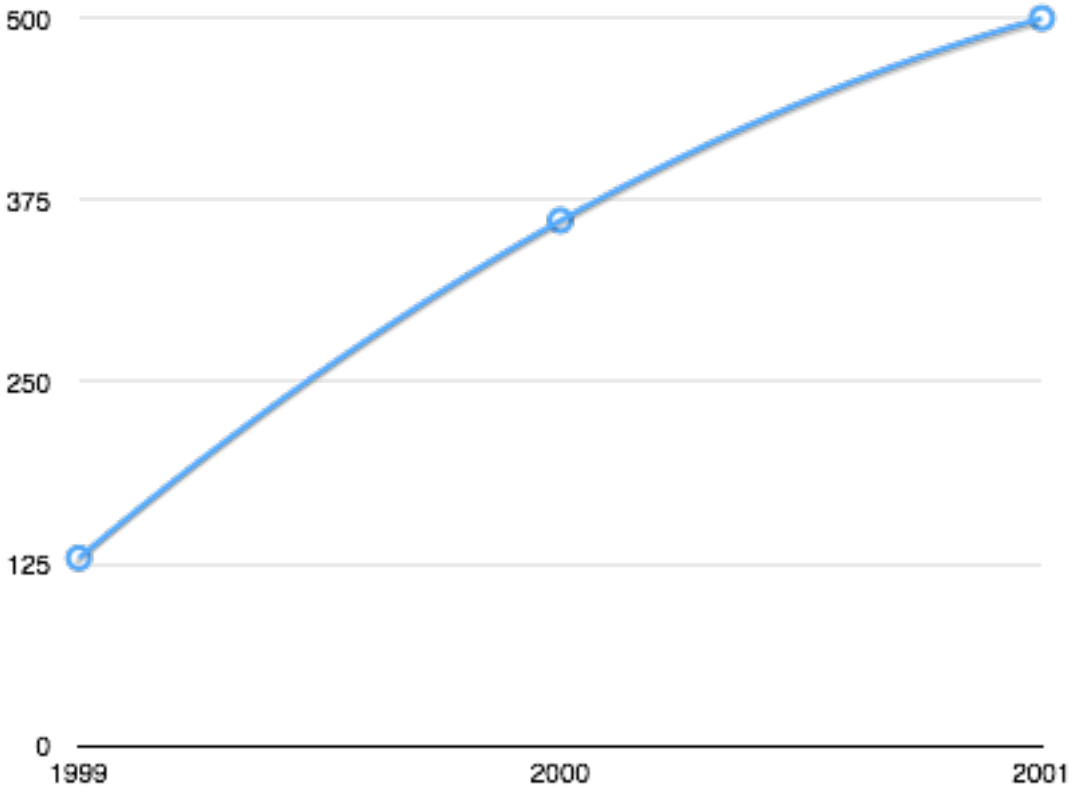


Slice Distribution

Tethne command-line (and TethneGUI) automatically generates a comma-separated values (CSV) file describing the number of records in each data slice. In your output folder look for a file called `[DATASET_ID]_sliceDistribution.csv`.



You can use your favorite spreadsheet software (e.g. Excel, Numbers, OpenOffice) to chart these data.



Python

Use the `tethne.data.DataCollection.slice()` method to slice your data.

```
>>> D.slice('date', 'time_period', window_size=1, cumulative=True)
```

Building the Coauthor Graph

Tethne will generate a graph using the `AU` field in your WoS data. See *Structure of the WoS Field-Tagged Data File* for more information about the fields available in a WoS datafile.

For now, we'll ignore data slicing and generate a single coauthor graph from the entire dataset using the `merged` option. Later on, we'll come back and use the data slicing to look at how the network evolves over time.

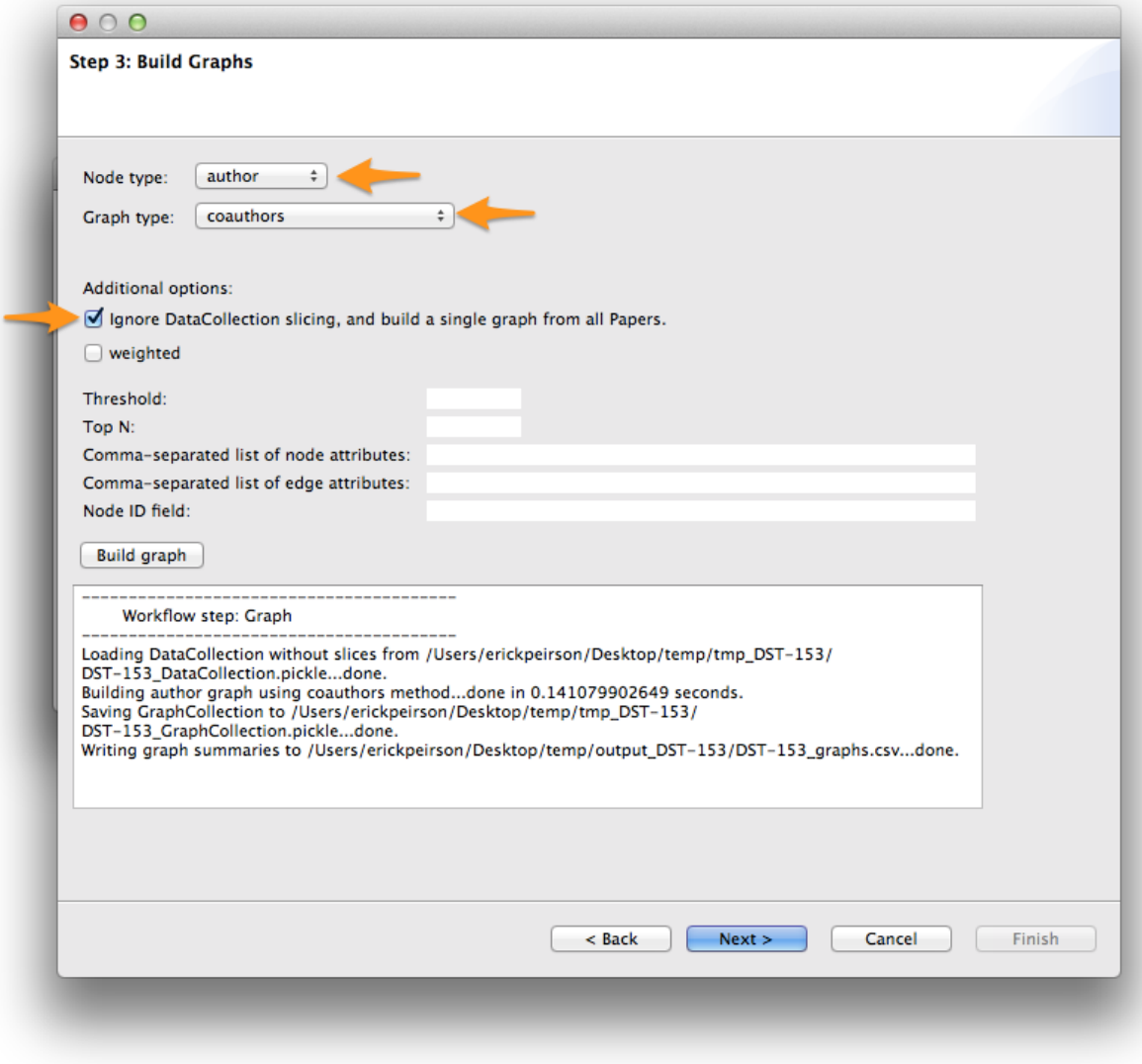
To generate a coauthorship network, we will tell Tethne to use authors for nodes, and use the `coauthors` graph type. For a complete list of graph types available in Tethne, see `networks`.

Command-line

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --graph --merged \
> --node-type=author --graph-type=coauthors
----------------------------------------
        Workflow step: Graph
----------------------------------------
Loading DataCollection without slices from /tmp/exampleID_DataCollection.pickle...done.
Building author graph using coauthors method...done in 0.144234895706 seconds.
Saving GraphCollection to /tmp/exampleID_GraphCollection.pickle...done.
Writing graph summaries to /Users/erickpeirson/exampleOutput/exampleID_graphs.csv...done.
```

TethneGUI

Select `author` from the `Node type` menu, and `coauthors` from the `Graph type` menu. Check the `Ignore DataCollection slicing` option, then click `Build graph`.



Once the graph is built, click `Next >`. For now, we'll skip the analysis step. Click `Next >` again to reach `Step 5: Write graph(s).`

Python

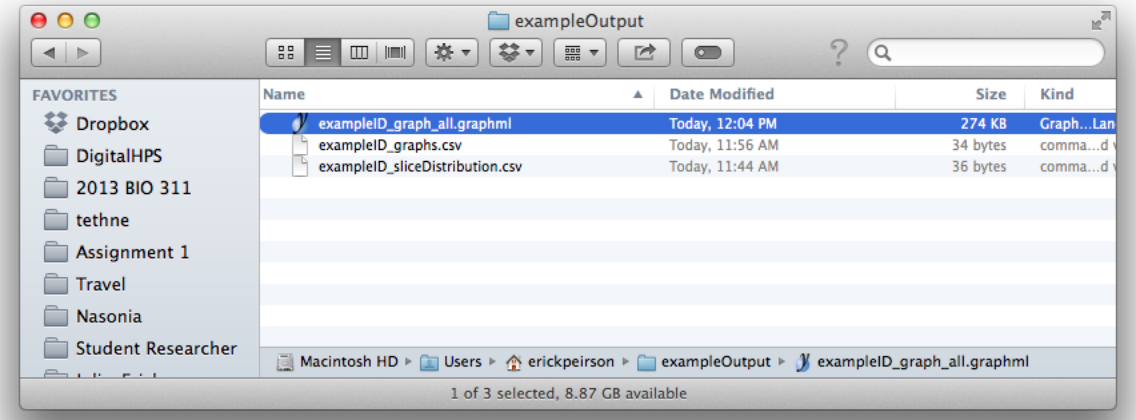
To generate a single graph from your `DataCollection`, call the `coauthors()` method directly from the `networks.authors` module.

```
>>> import tethne.networks as nt
>>> ca_graph = nt.authors.coauthors(D.papers())
```

Write the Graph to GraphML

GraphML is a widely-used static network data format. We will write our graph to GraphML for visualization in Cytoscape.

This step should generate a file in your output folder called `[DATASET_ID]_graph_all.graphml`.



Command-line

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --write --write-format graphml
----------------------------------------
        Workflow step: Write
----------------------------------------
Loading GraphCollection from /tmp/exampleID_GraphCollection.pickle...done.
Writing graphs to /Users/erickpeirson/exampleOutput with format graphml...done.
```

TethneGUI

Select `graphml` from the `Output format for graph(s)` menu, then click `Write graph(s)`.

Python

Use the `to_graphml()` method in `writers.collection` to create a GraphML data file.

```
>>> import tethne.writers as wr
>>> wr.graph.to_graphml(ca_graph, "[OUTPUT_PATH]")
```

`[OUTPUT_PATH]` should be a path to the GraphML file that Tethne will create.

Visualizing the Merged Network in Cytoscape

Cytoscape was developed in 2002, with funding from the National Instute of General Medical Sciences and the National Resource for Network Biology. The primary user base is the biomedical research community, especially systems biologists who study gene or protein interaction networks and pathways.

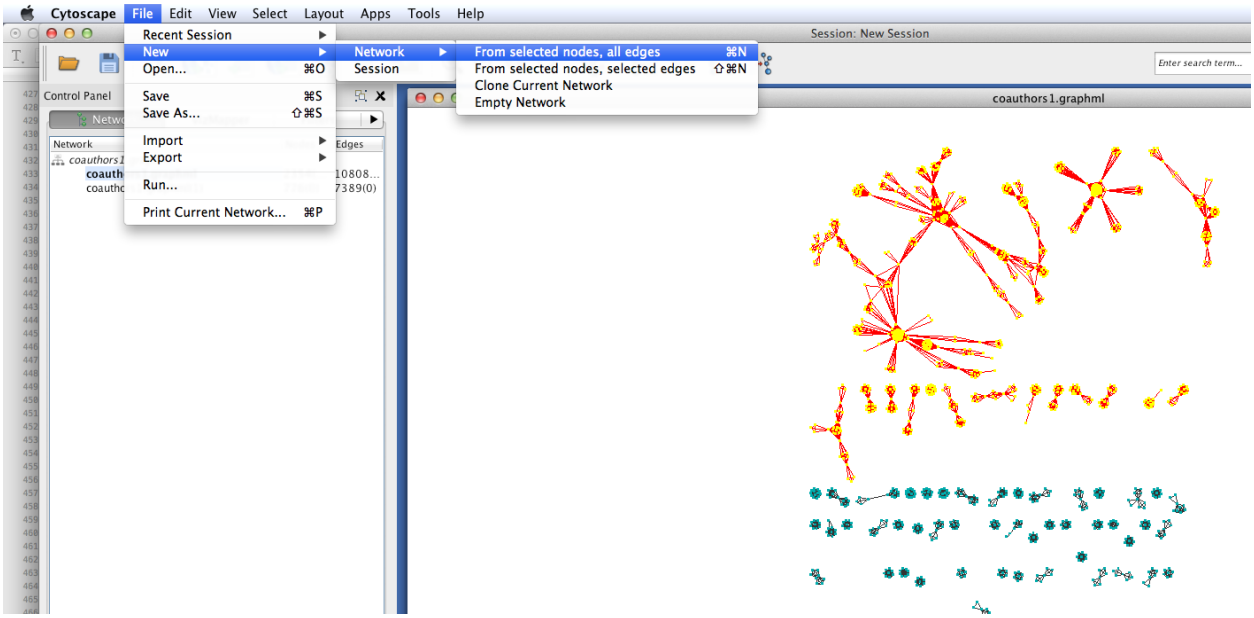
You can download Cytoscape 3 from url{http://www.cytoscape.org}. This tutorial assumes that you are using Cytoscape 3.0.2.

Import

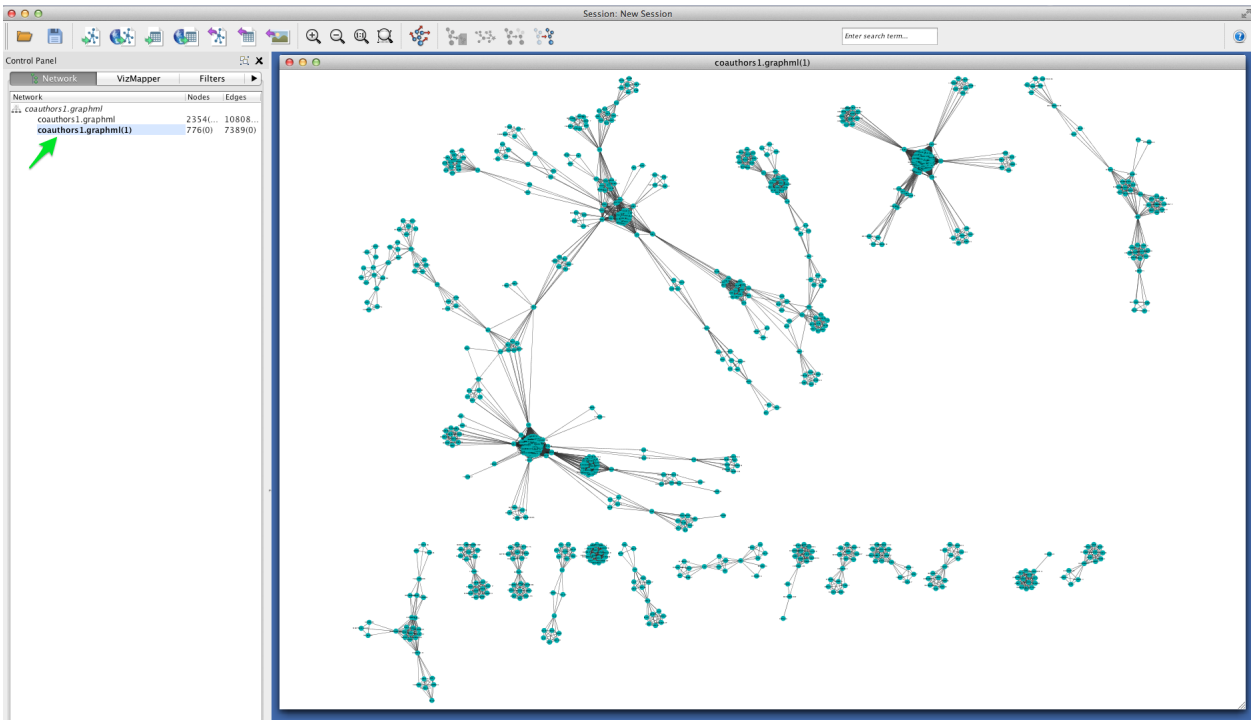
In Cytoscape, import your network by selecting `File > Import > Network > From file...` and selecting the GraphML file generated by Tethne in your output directory.

Apply a Force Directed layout by selecting `Layout > Prefuse Force Directed Layout`.

Coauthorship networks are usually comprised of a very large connected component, and many very small components. For convenience, we will only look at the few largest components. Select the largest connected components (click and drag to create a selection box). Then create a new network with those selected components: select `File > New > Networks > From selected nodes, all edges`.



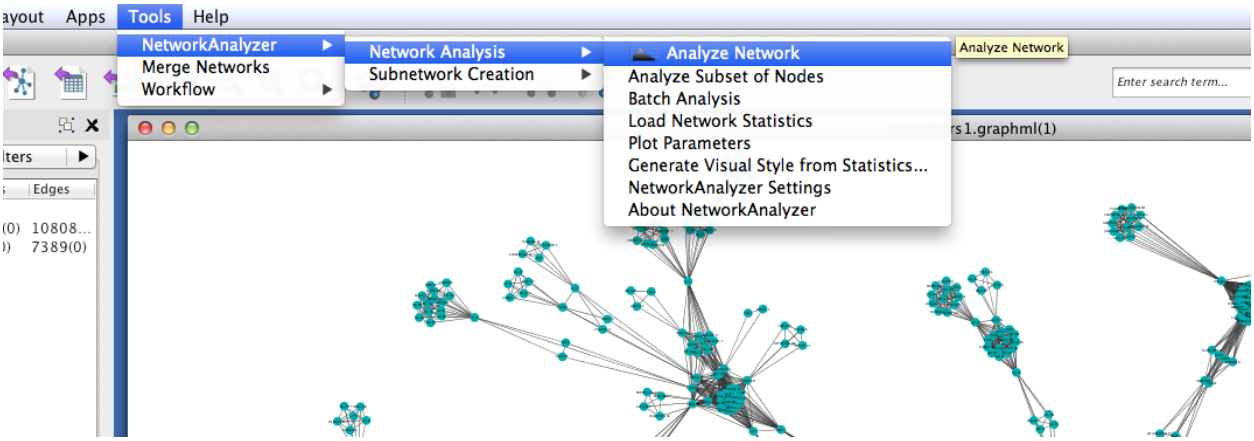
You should now see a new graph in its own viewing window, containing only the components that you selected.



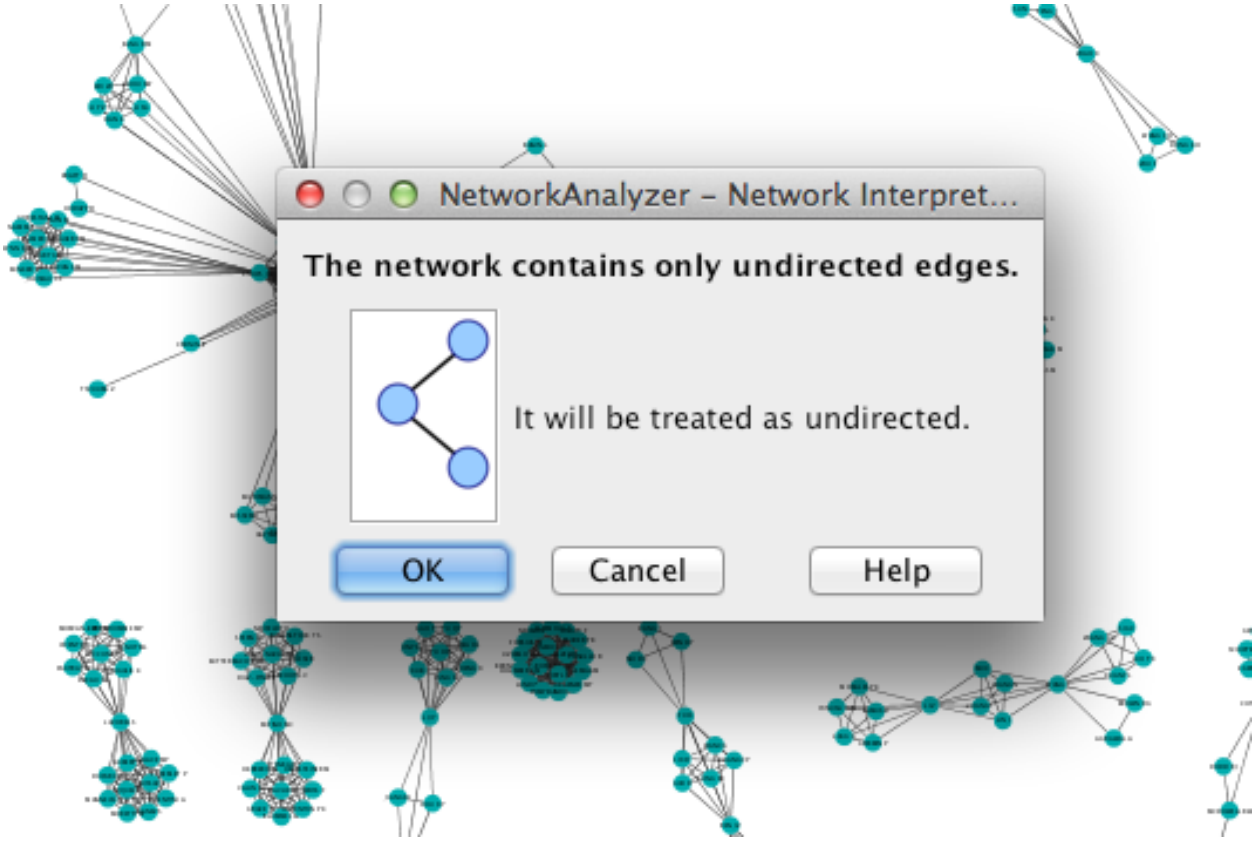
Betweenness Centrality

This coauthorship network is clearly very modular: there are dense clusters connected by a few linking nodes that occupy sparse areas of the graph (so-called "structural holes"). We can identify the structurally most-significant actors by their "betweenness centrality." Formally, betweenness centrality is a measure of the number of shortest paths that pass through a particular node.

Run Cytoscape's network-analysis algorithm. Go to `Tools > NetworkAnalyzer > Network Analysis > Analyze Network`.



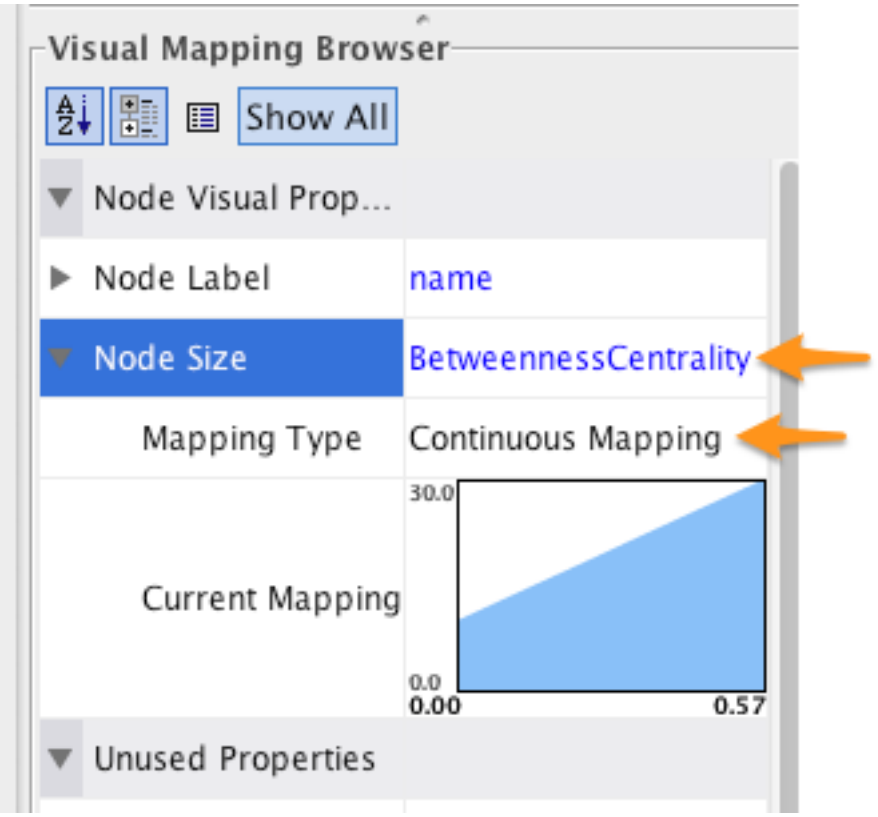
Cytoscape may ask you whether to interpret the network as directed or undirected. A coauthorship network is always undirected, since coauthorship is a symmetric relationship.



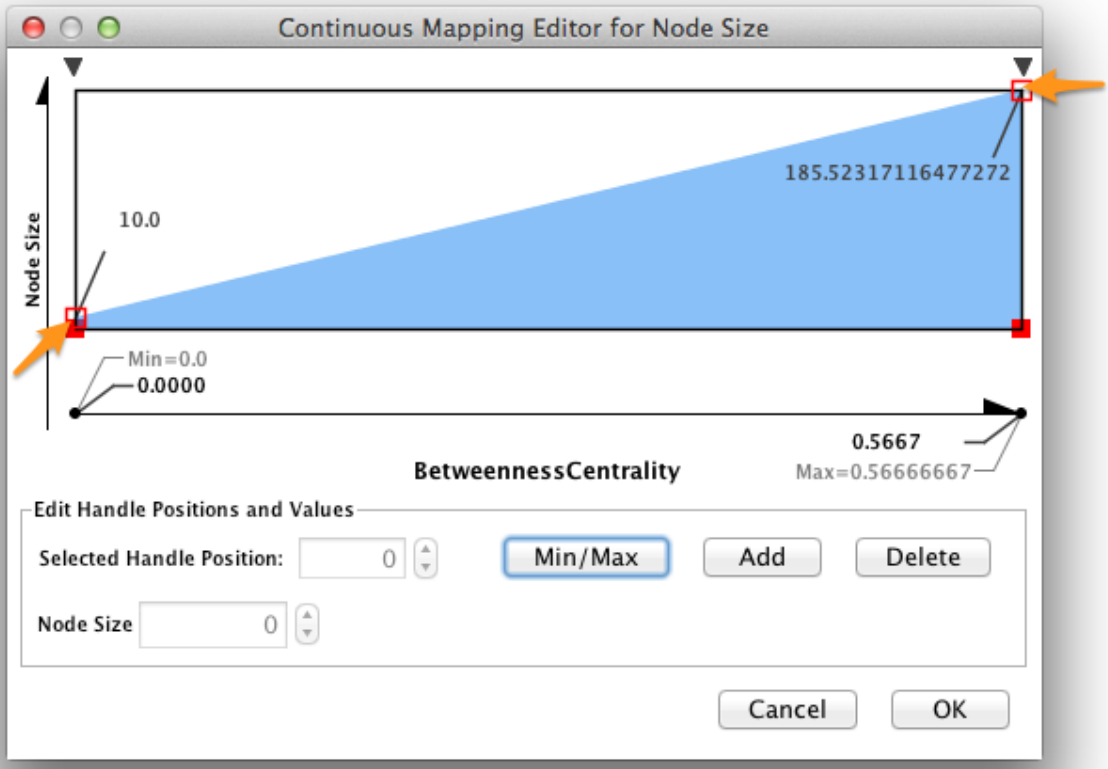
Once network analysis is complete, a window titled `Results Panel` will appear. Close this window.

To visualize the betweenness centrality of each node, create a new visual mapping.

1. Go to the VizMapper tab, in the left part of the Cytoscape workspace.
2. Find `Node Size` in the unused visual properties, and double-click to move it to the `Node Visual Properties` list.
3. Click in the area to the right of `Node Size` and select `BetweennessCentrality`.
4. Click in the area to the right of `Mapping Type` and select `Continuous Mapping`.

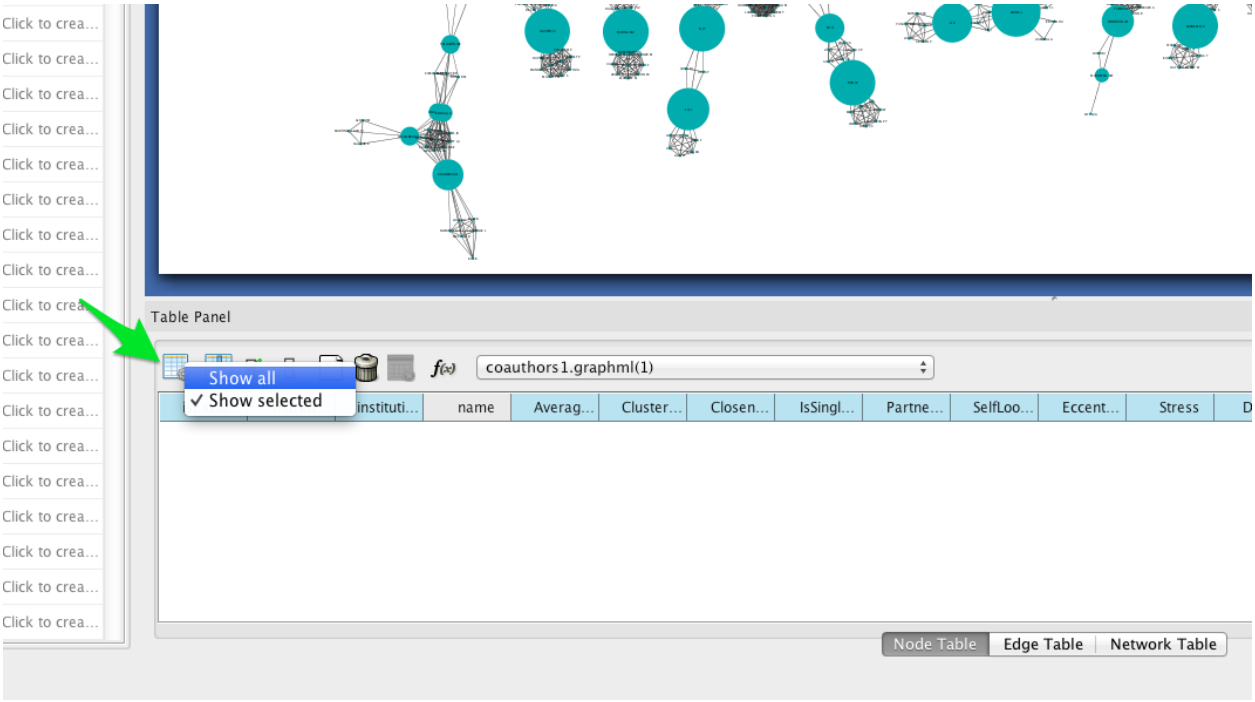


To change the size - centrality mapping function, double-click on the figure to the right of `Curent Mapping`, and drag the red open boxes up and down to change the angle of the function.

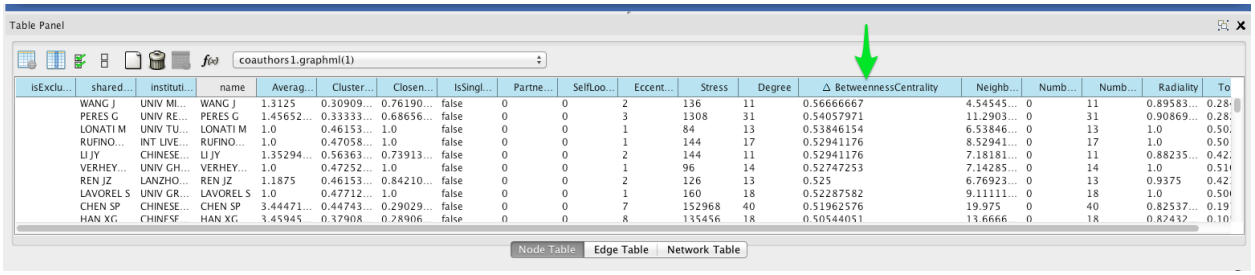


The largest nodes are the most central nodes in their respective connected components. These are the nodes most responsible for connecting disparate clusters in the network.

To see a list of the most central nodes, set the Table Panel to show all nodes.

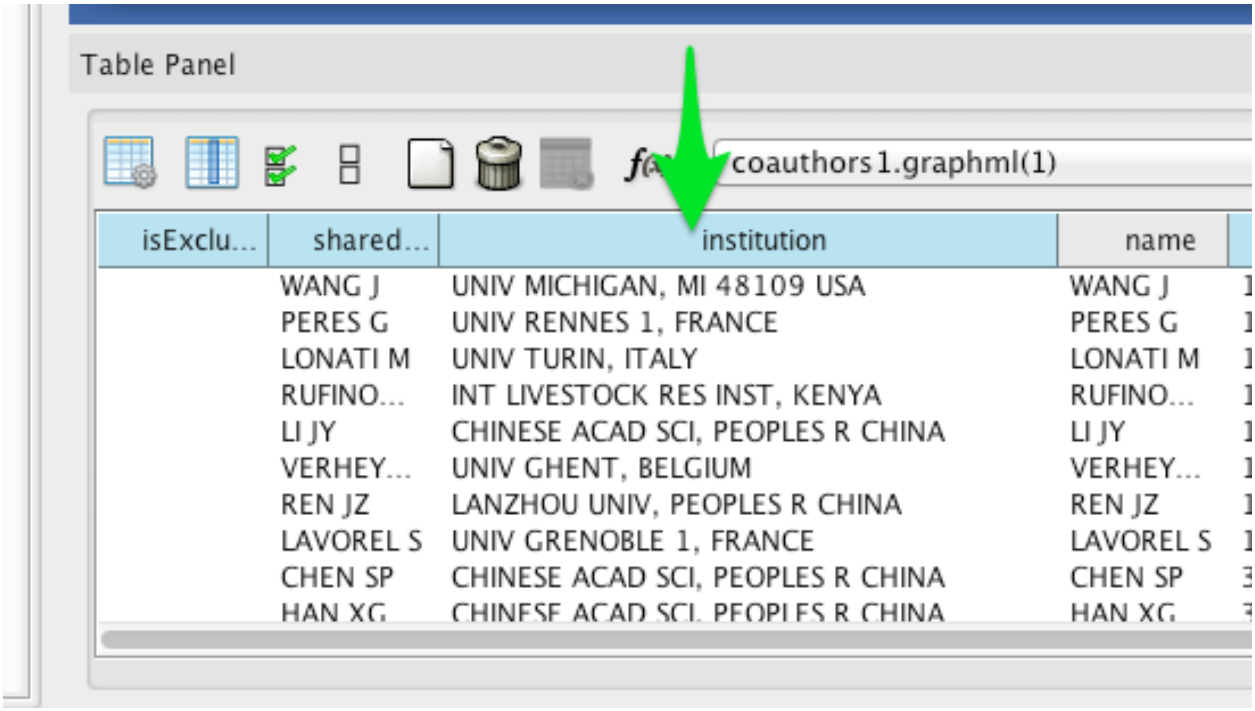


Then sort by betweenness centrality by clicking on the column header in the Node Table (you may have to click twice to sort in descending order).



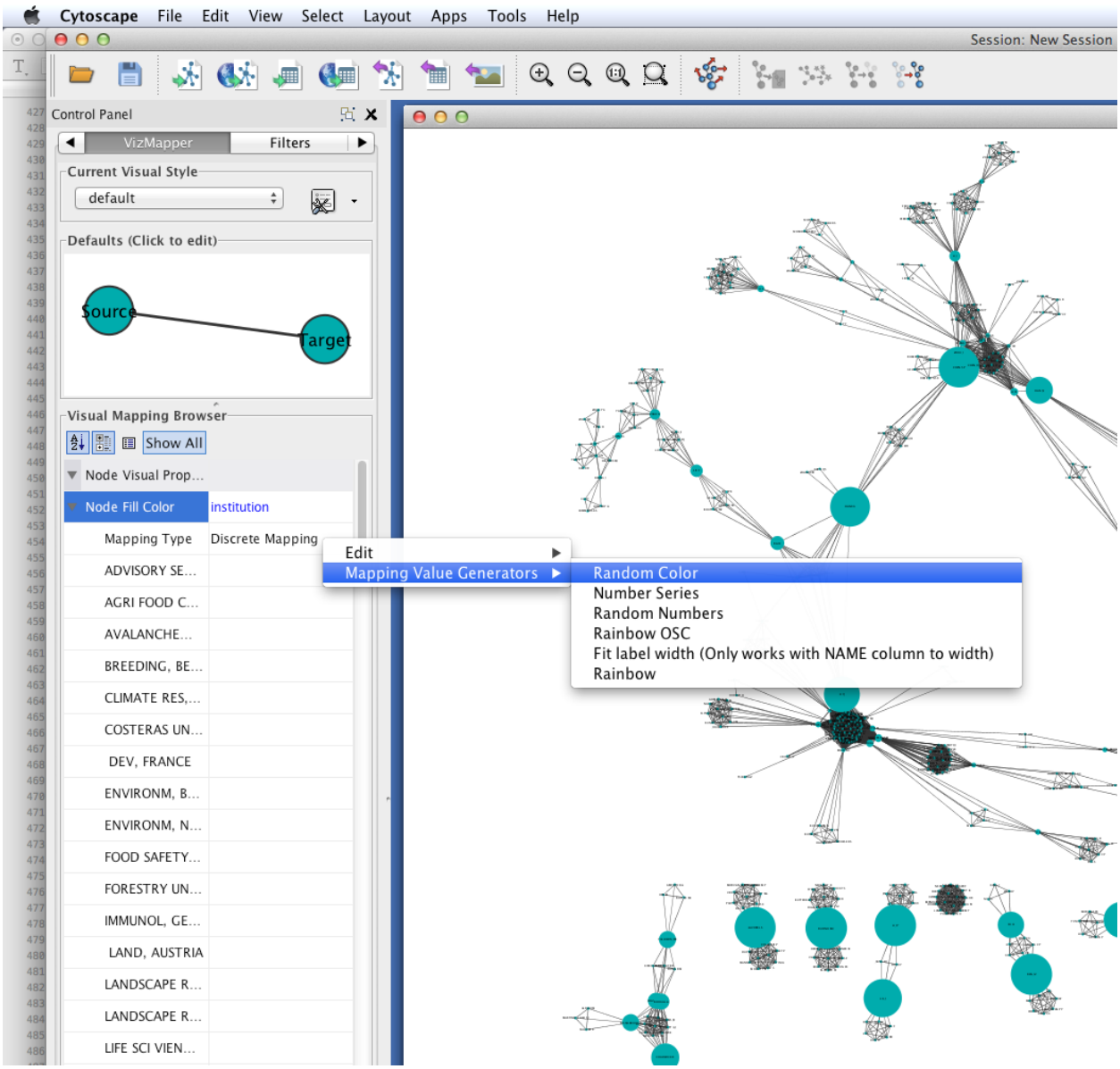
Institutional affiliation

Wherever possible, Tethne includes institutional affiliations for authors as node attributes. You should see institutions listed in the Node Table.

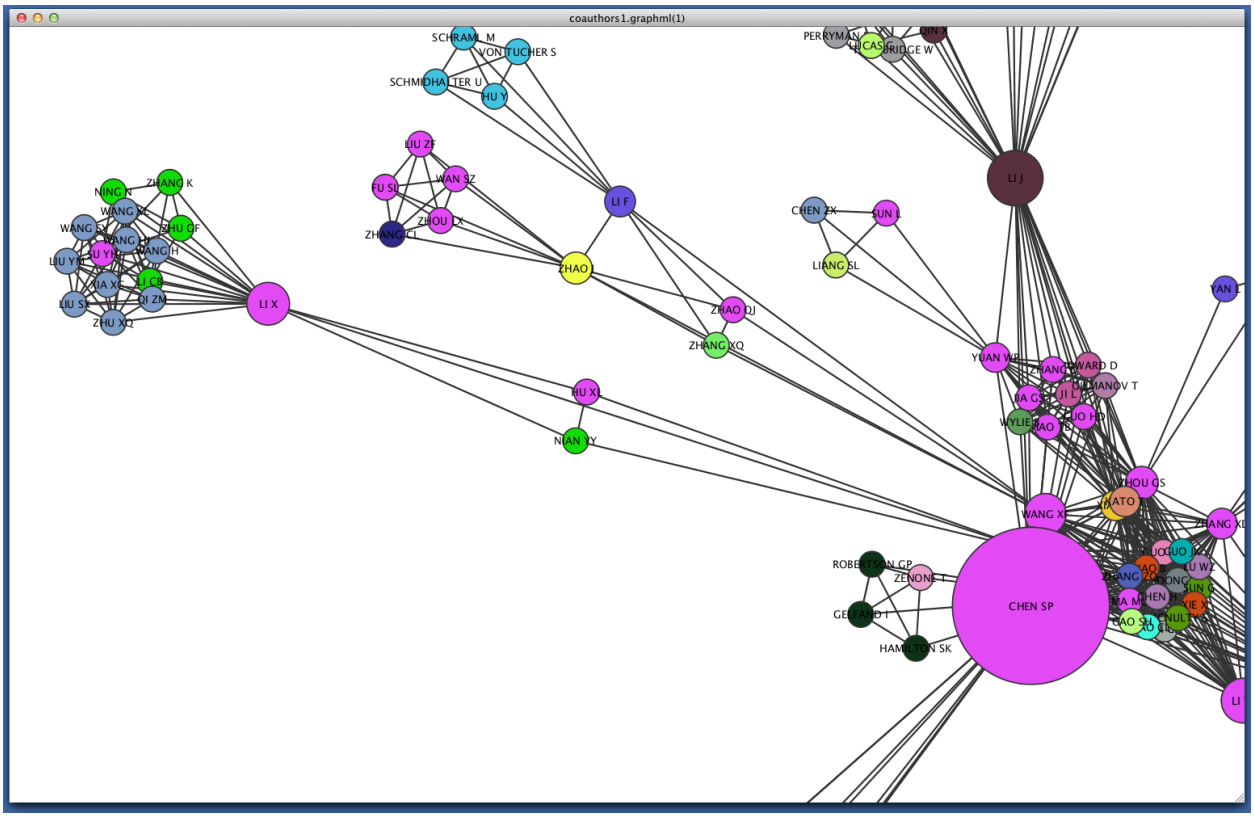


Create a visual mapping for institutional affiliation.

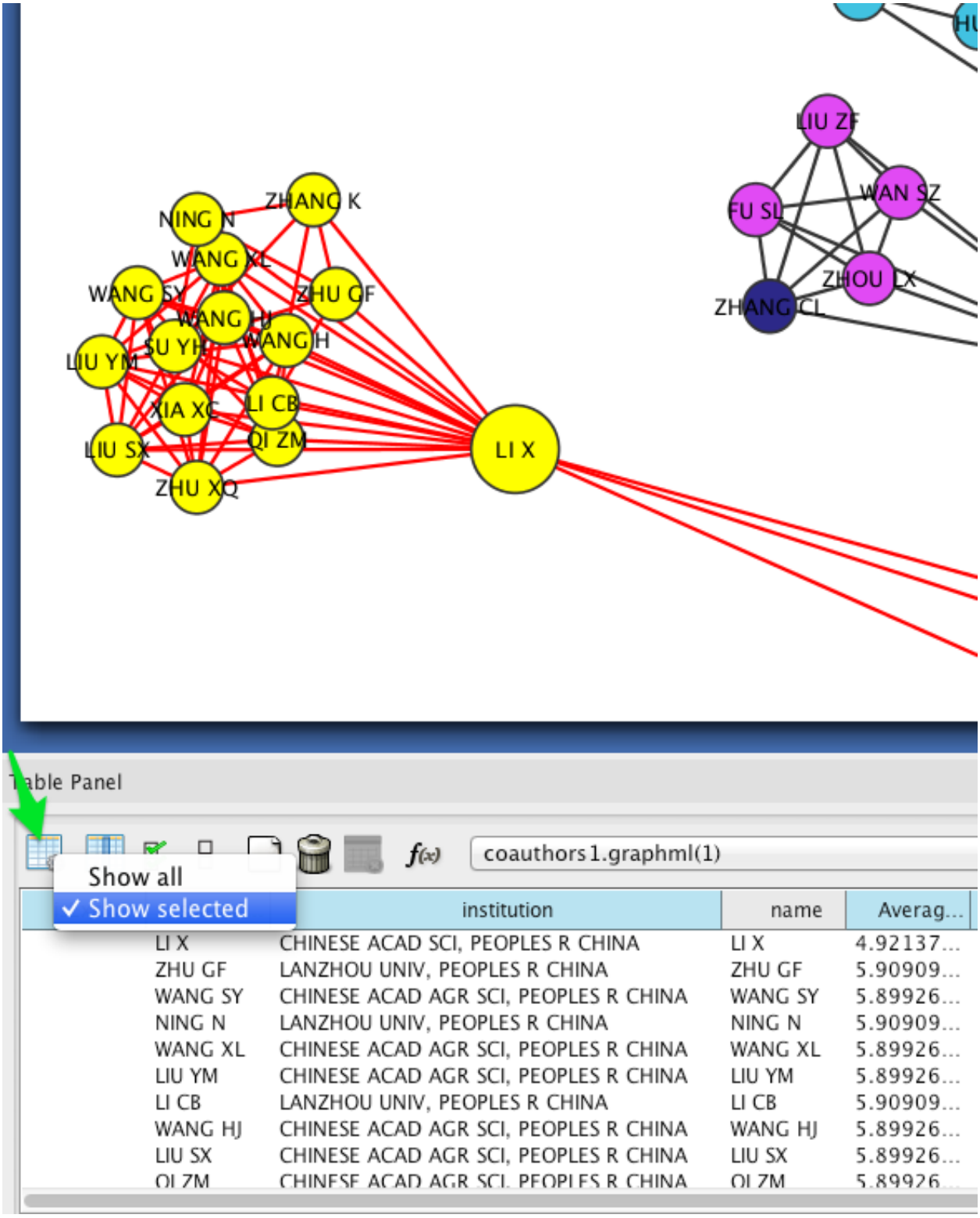
1. Go to the VizMapper.
2. Find `Node Fill Color` in the unused visual properties, and double-click to activate.
3. Click to the right of ‘‘Node Fill Color” and select ‘‘institution’‘.
4. Set the ‘‘Mapping Type” to ‘‘Discrete Mapping.” A list of institutions should appear below ‘‘Mapping Type.’‘
5. Right-click on `Discrete Mapping", and select ``Mapping Value Generators > Random Color`.



Each node should now be colored according to its institutional affiliation. Inspecting the network yields an immediate impression of whether coauthorship clusters are due to affiliation with the same institution.

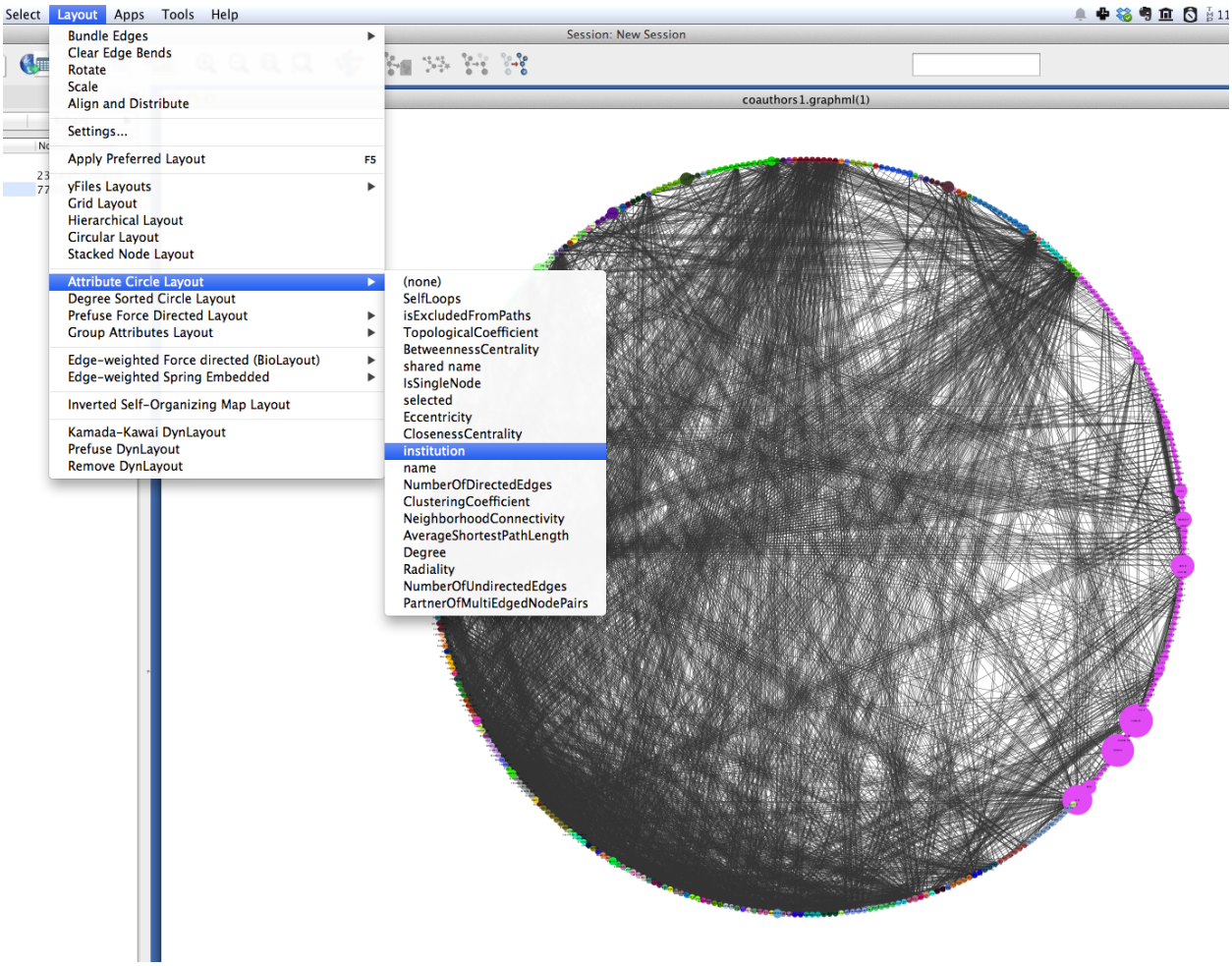


Since some institutions may be colored quite similarly, select a cluster to view the specific institutional affiliation of each node. You may need to set the Node Table to `show selected` rather than `show all`.

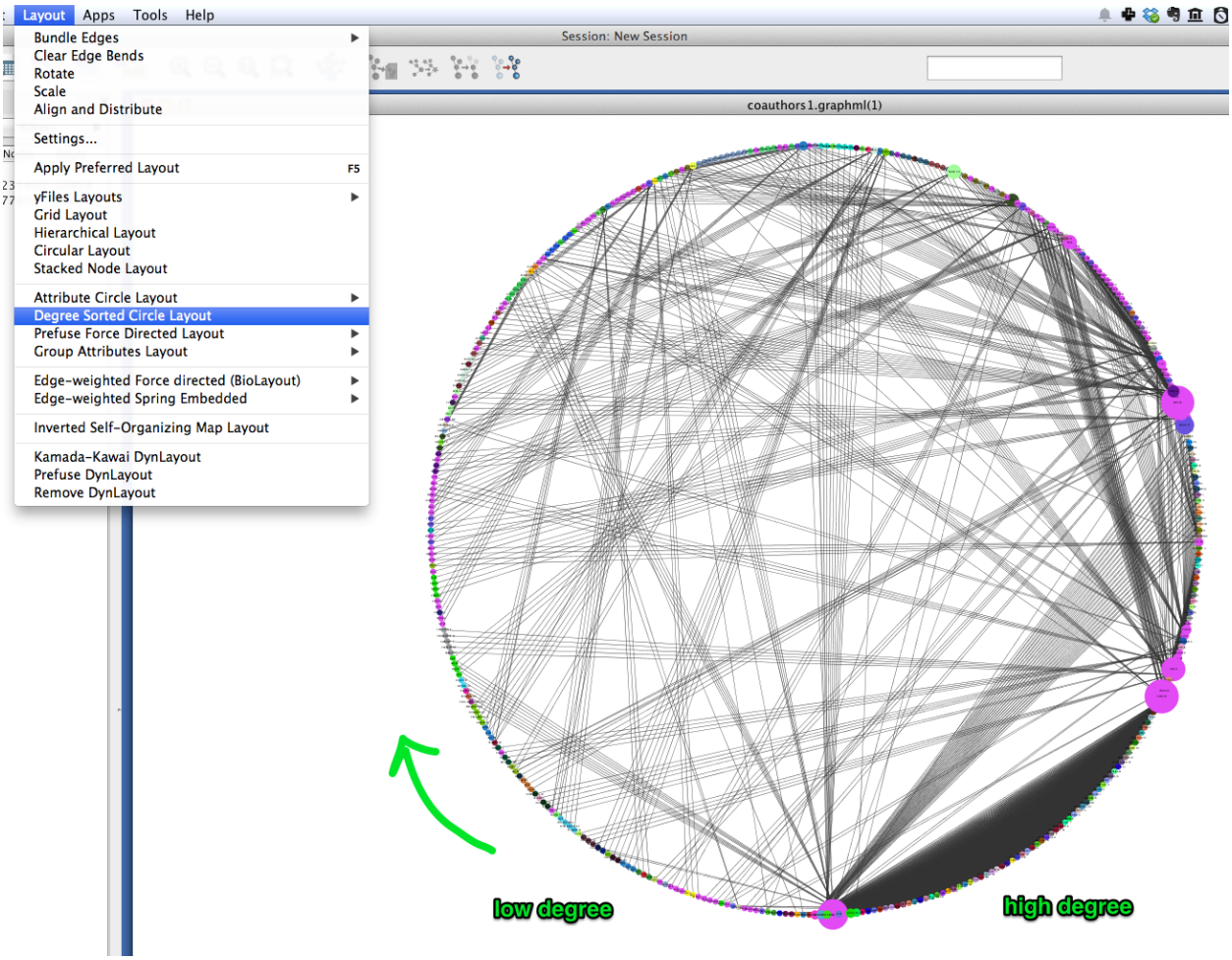


	institution	name	Averag...
LI X	CHINESE ACAD SCI, PEOPLES R CHINA	LI X	4.92137...
ZHU GF	LANZHOU UNIV, PEOPLES R CHINA	ZHU GF	5.90909...
WANG SY	CHINESE ACAD AGR SCI, PEOPLES R CHINA	WANG SY	5.89926...
NING N	LANZHOU UNIV, PEOPLES R CHINA	NING N	5.90909...
WANG XL	CHINESE ACAD AGR SCI, PEOPLES R CHINA	WANG XL	5.89926...
LIU YM	CHINESE ACAD AGR SCI, PEOPLES R CHINA	LIU YM	5.89926...
LI CB	LANZHOU UNIV, PEOPLES R CHINA	LI CB	5.90909...
WANG HJ	CHINESE ACAD AGR SCI, PEOPLES R CHINA	WANG HJ	5.89926...
LIU SX	CHINESE ACAD AGR SCI, PEOPLES R CHINA	LIU SX	5.89926...
QI ZM	CHINESE ACAD AGR SCI, PEOPLES R CHINA	QI ZM	5.89926...

Circular layouts can also yield some insights into connectivity between different institutions. In the menu bar, select `Layout > Attribute Circle Layout > institution`. This should arrange the nodes in each connected component in a circle. Nodes that are affiliated with the same institution should be adjacent to each other, so that the circumference of each circle can be divided into regions that correspond to single institutions. Edges crossing from one region to another should give a visual impression of the magnitude of linkages between institutions.



A similar layout, the `Degree Sorted Circle` layout, can yield more information about the structure of the network. As the name suggests, this layout arranges nodes in ascending order of degree (the number of links that each node has with other nodes in the network). The lowest-degree nodes begin just west of due-south, and degree increases clockwise around the circle so that the highest-degree nodes are just east of due-south. In the network depicted below, there is extremely dense connectivity among the highest-degree nodes, while the rest of the graph is sparse by comparison. In other words, the most well-connected nodes are all highly connected to each other. This may be due in part to papers with a very large number of authors.



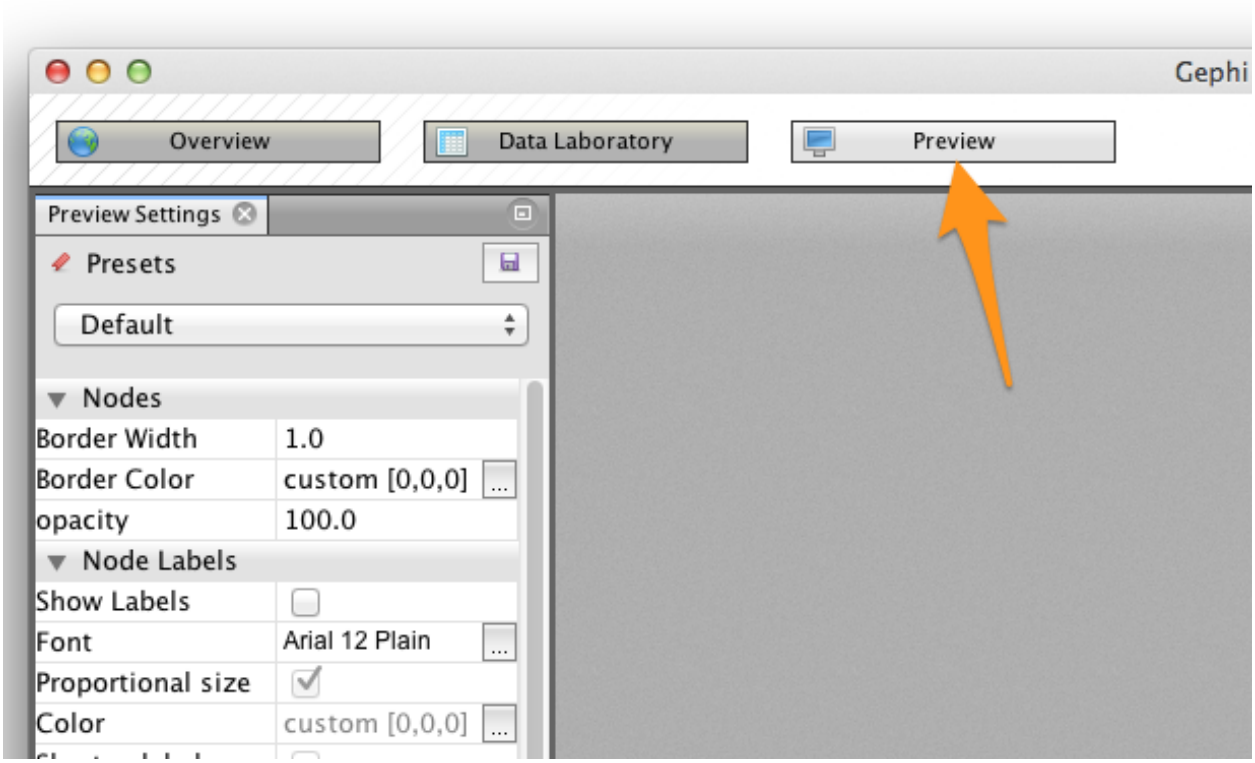
To export an image of your network, select `File > Export > Current Network View as Graphics`, and follow the prompts to save your image.

Inter-institutional Collaboration in Gephi

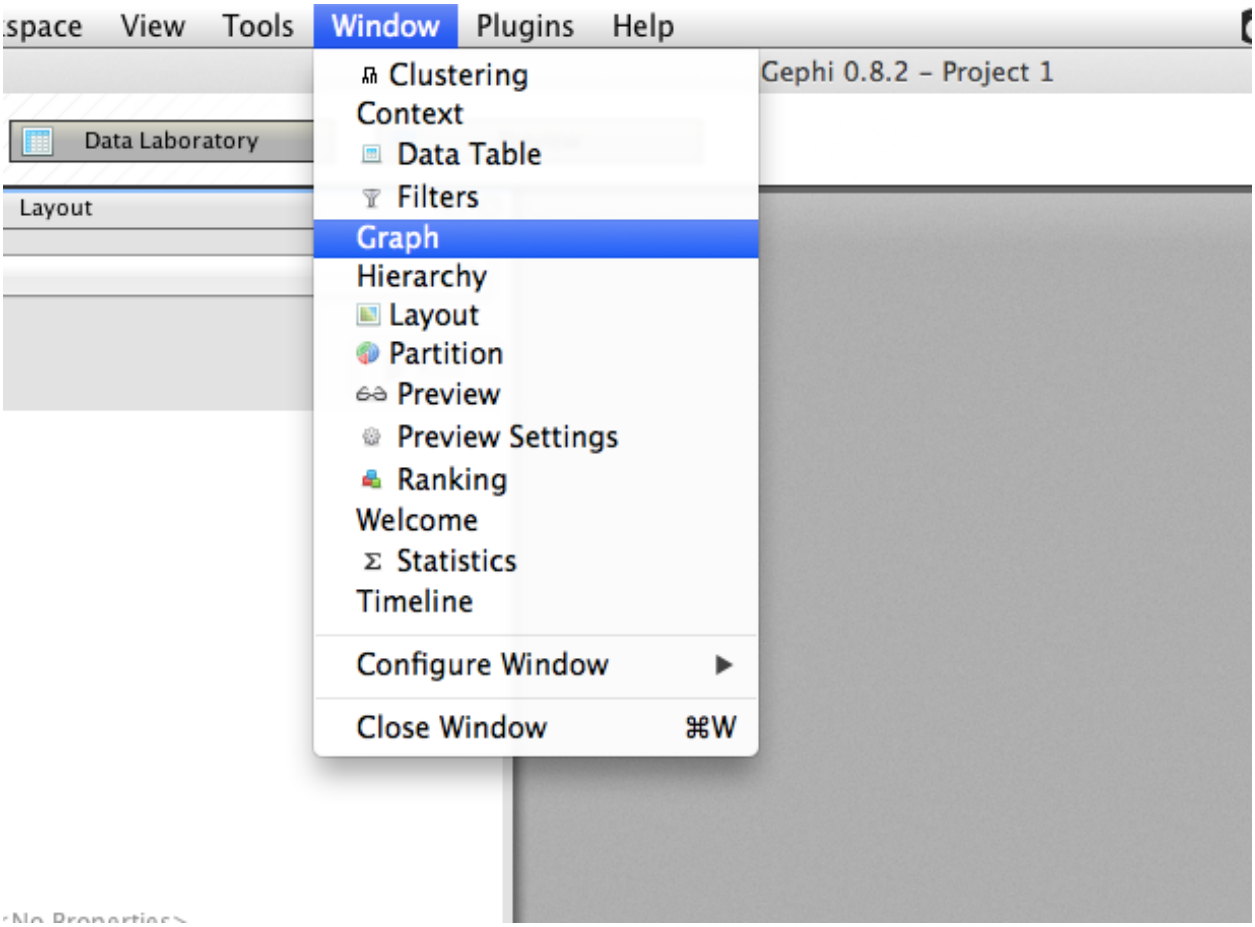
Gephi provides additional tools for analyzing coauthorship networks. In this section, we'll use Gephi to generate an inter-institutional collaboration network using your coauthorship network. That is, we will mash authors from the same institutions together into institutional nodes, and combine coauthorship edges so that we can see the magnitude of coauthorship activity between different institutions.

Import & visualize

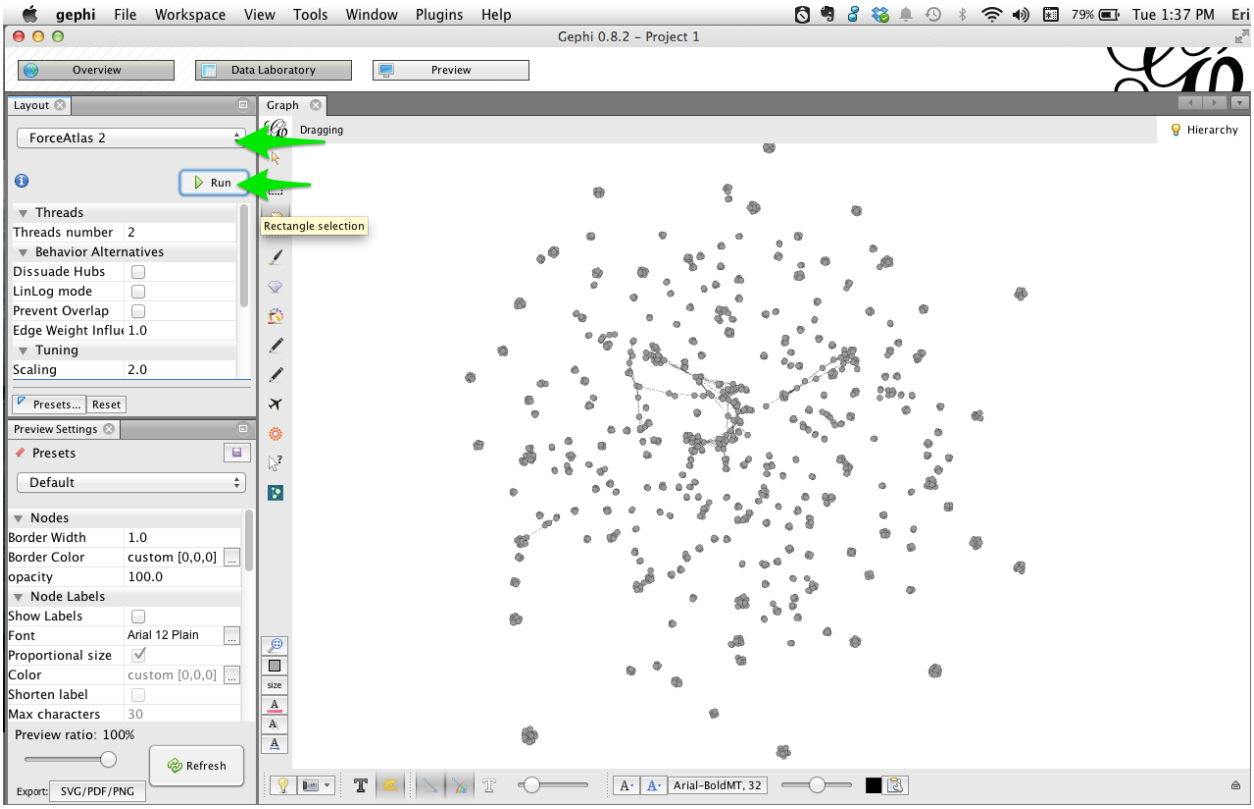
1. In Gephi, select `File > Open...` and select your GraphML network file.
2. Click on the `Preview` tab.



3. Open the `Graph` window: select `Window > Graph`.

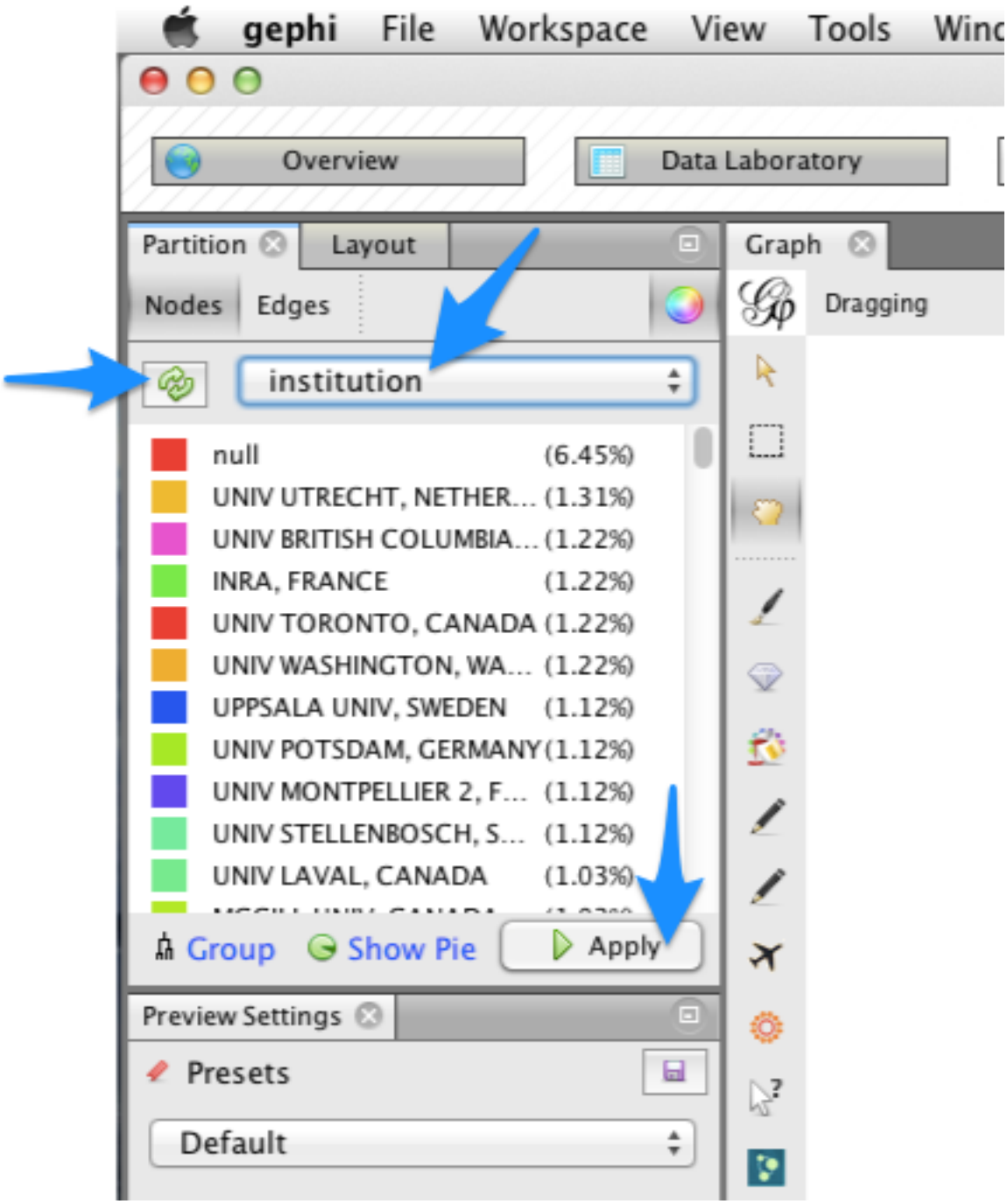


4. Open the `Layout` window: select `Window > Layout`.
5. In the `Layout` window, select the `Force Atlast 2 layout`, then click `Run`. After a few seconds the graph should be spread out; click `Stop`.

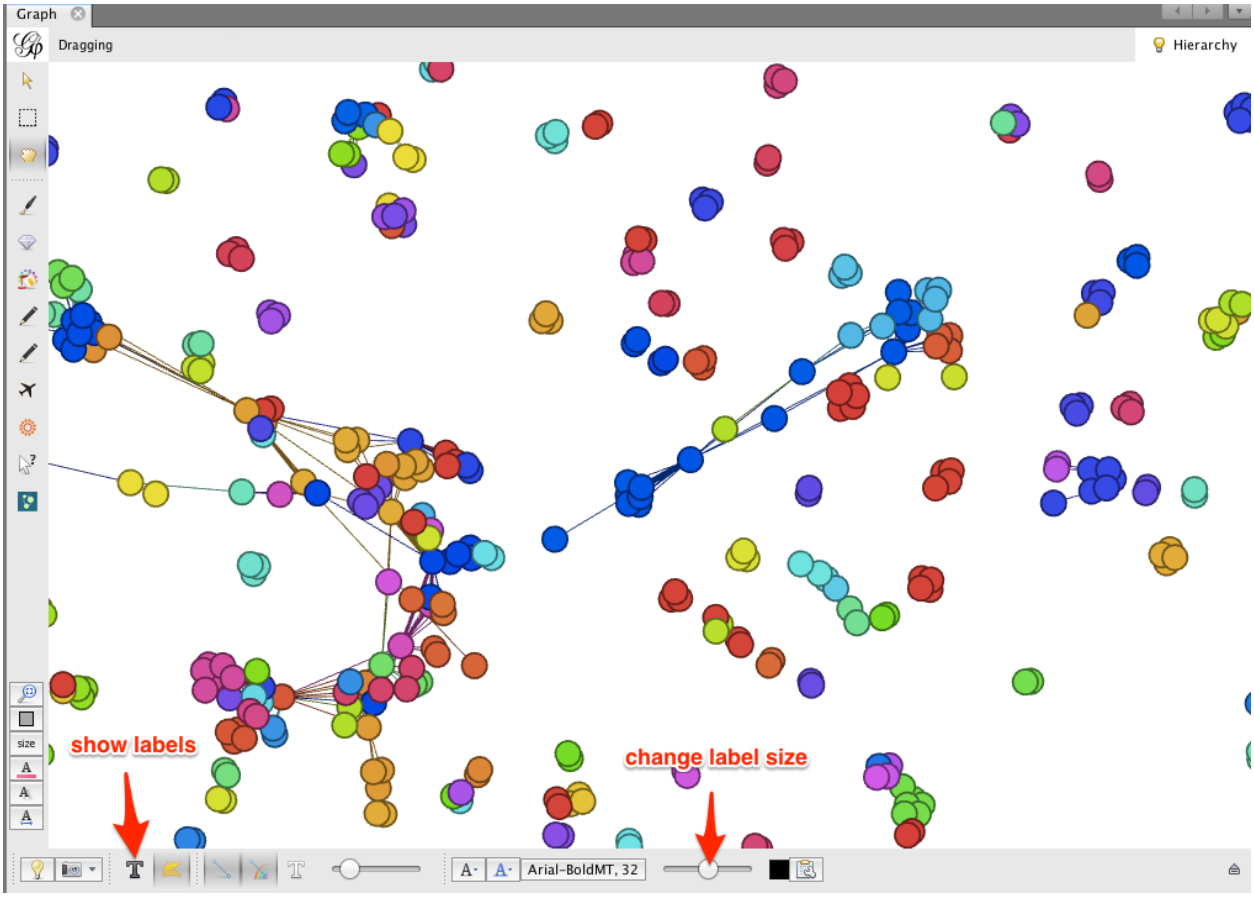


Partition by institution

1. Open the `Partition` window: select `Window > Partition`. You may need to drag the window to the left-hand area of the Gephi workspace.
2. In the `Partition` window, you should be on the `Nodes` tab by default. Click the green "refresh" button, then select "institution" from the drop-down menu. You should see a list of all institutions.
3. To color nodes by institution, click the `Apply` button.



Zooming in on the network, you'll notice that some clusters of nodes are comprised of one or a few colors, while other clusters are quite mixed. Just as in Cytoscape, this gives a visual impression of which research communities involve inter-institutional collaborations, and which are more internal to a particular institution.



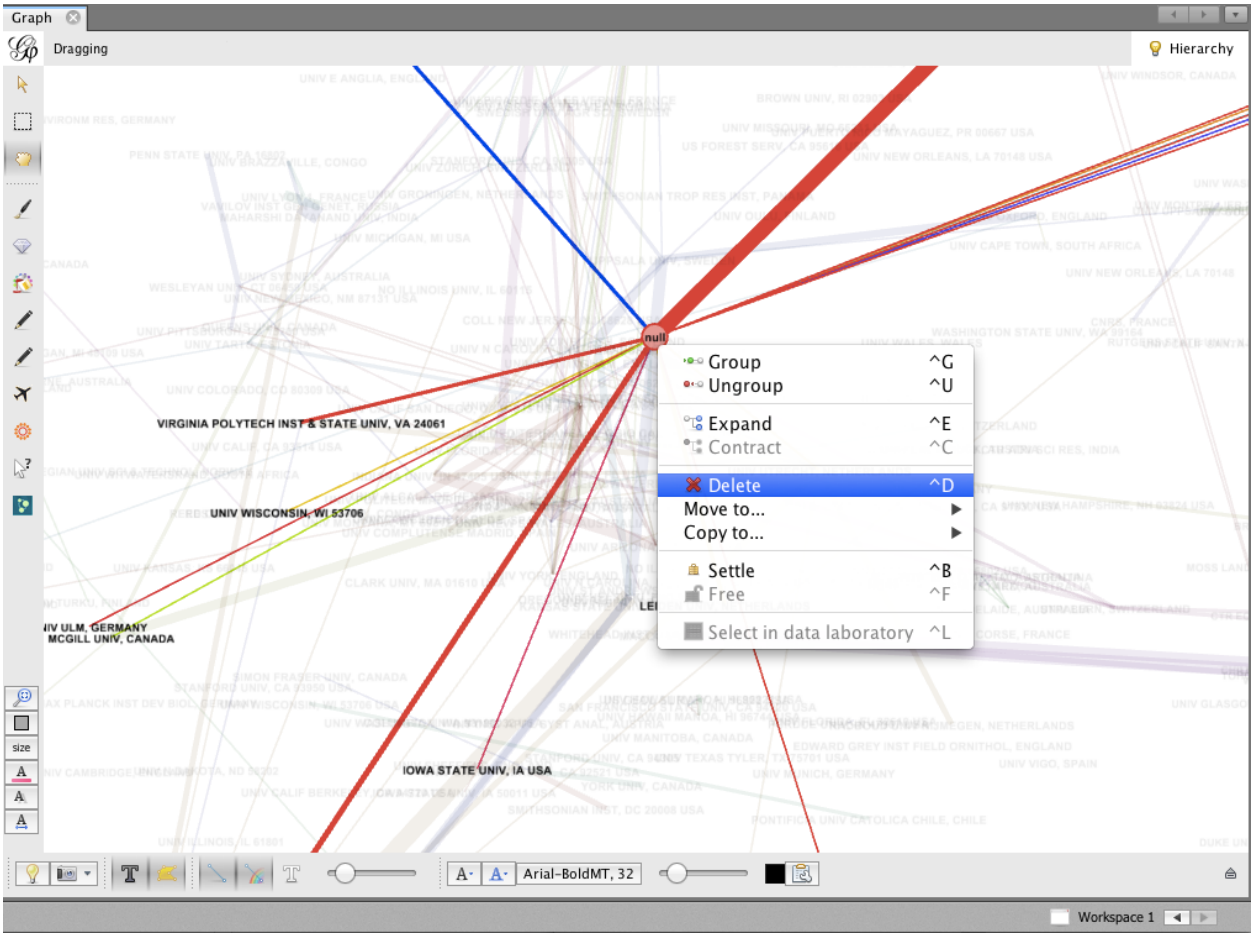
Gephi makes it easy to collapse individual author nodes into nodes corresponding to their institutions. Cytoscape has this feature as well, but not all of the bugs are completely worked out.

To group authors together into their respective institutions, click the `Group` button in the `Partition` window.

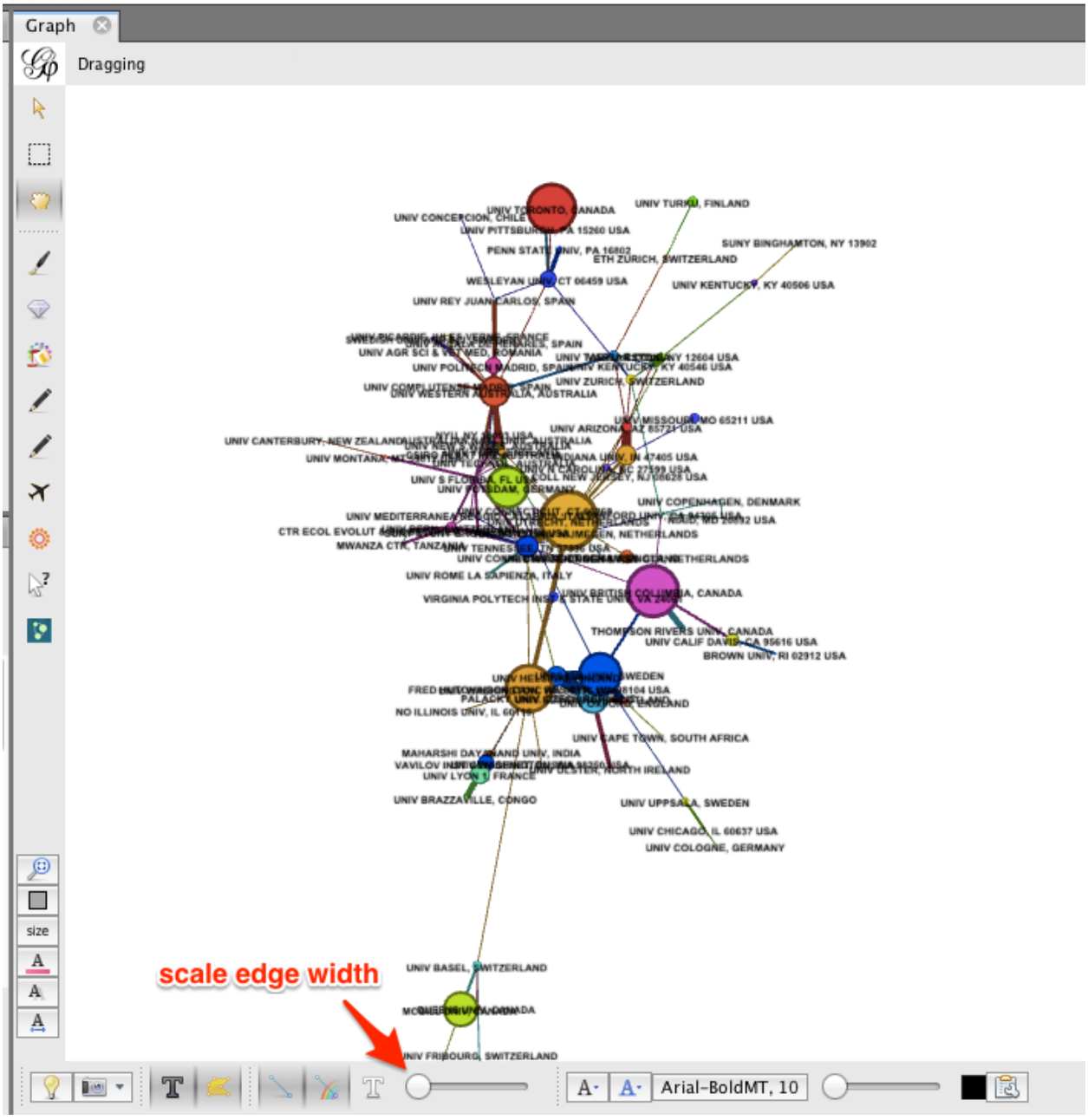
Click on the dark **T** button in the lower left corner to show node labels, and use the right-hand slider at the bottom of the Graph window to make the labels smaller or larger.

The result may look a bit messy. There are a few things to notice:

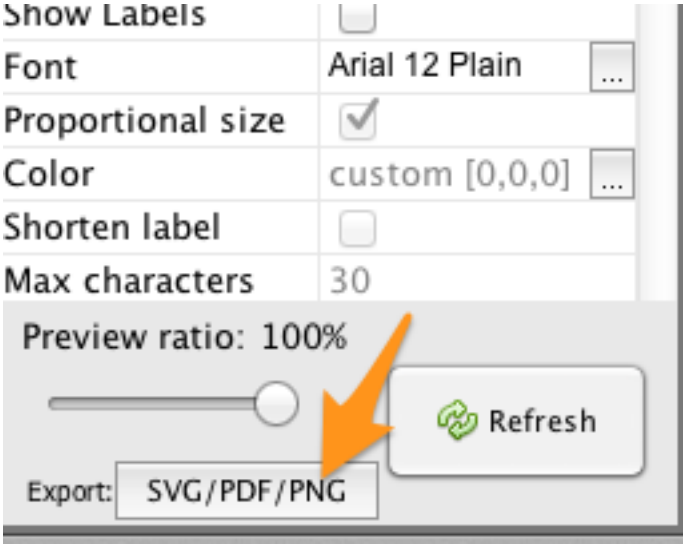
- The edges between authors have been pooled into edges between institutions. The edge weight indicates the number of coauthorship relationships between a pair of institutions.
- The biggest node is called `null`. This represents all of the authors for which no institutional information was available. You may wish to delete this node; right-click on the node and select `Delete`. When prompted, click `Yes`.



To re-layout the network, go back to the `Layout` tab, and run the layout algorithm again. You may notice that the network contracts rapidly. You may find it useful to reduce the edge width and zoom in, to achieve a nice node-size : edge-weight ratio.



To save an image of your network, click the `SVG/PDF/PNG` button in the lower-left corner of the Gephi workspace.



Coauthorship network evolution

This section describes how to generate a dynamic network with Tethne, and visualize that network in Cytoscape. Dynamic networks allow us to go beyond analyzing the final structure of a network, and ask how the structure of a network changes over time. In this case, we will use a dynamic network to see how a coauthorship network grows over time.

Since we used the `--cumulative` option when slicing our data, our dynamic network will only involve the addition of nodes and edges: older coauthorship relationships will not "expire."

A seemingly ubiquitous property of social networks is that they tend to be "scale-free". That is, the degree distribution follows a power-law: there are a few very highly-connected actors, and a very large number of poorly-connected actors. The intuitive interpretation of this behavior is that "the rich get richer." In other words, if you're already popular then you're more likely to make new friends.

In this tutorial, we will visualize the impact of degree centrality on edge acquisition by using the `analyze.collection.attachment_probability()` algorithm in Tethne.

Command-line

Run the `graph` step again, but this time remove the `--merged` flag. This will create a separate graph from each of the data subsets created in the `slice` step.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --graph --node-type=author --graph-type=co
----------------------------------------
        Workflow step: Graph
----------------------------------------
Loading DataCollection with slices from /tmp/exampleID_DataCollection_sliced.pickle...done.
Using first slice in DataCollection: date.
Building author graph using coauthors method...done in 0.291323900223 seconds.
Saving GraphCollection to /tmp/exampleID_GraphCollection.pickle...done.
Writing graph summaries to /Users/erickpeirson/exampleOutput/exampleID_graphs.csv...done.
```

Use the `-A attachment_probability` argument in the `--analyze` step.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --analyze -A attachment_probability
----------------------------------------
```

```
        Workflow step: Analyze
---------------------------------------
Loading GraphCollection from /tmp/exampleID_GraphCollection.pickle...done.
Analyzing GraphCollection with attachment_probability...done.
Writing graph analysis results to /Users/erickpeirson/exampleOutput/exampleID_attachment_probability_
Saving GraphCollection to /tmp/exampleID_GraphCollection.pickle...done.
```

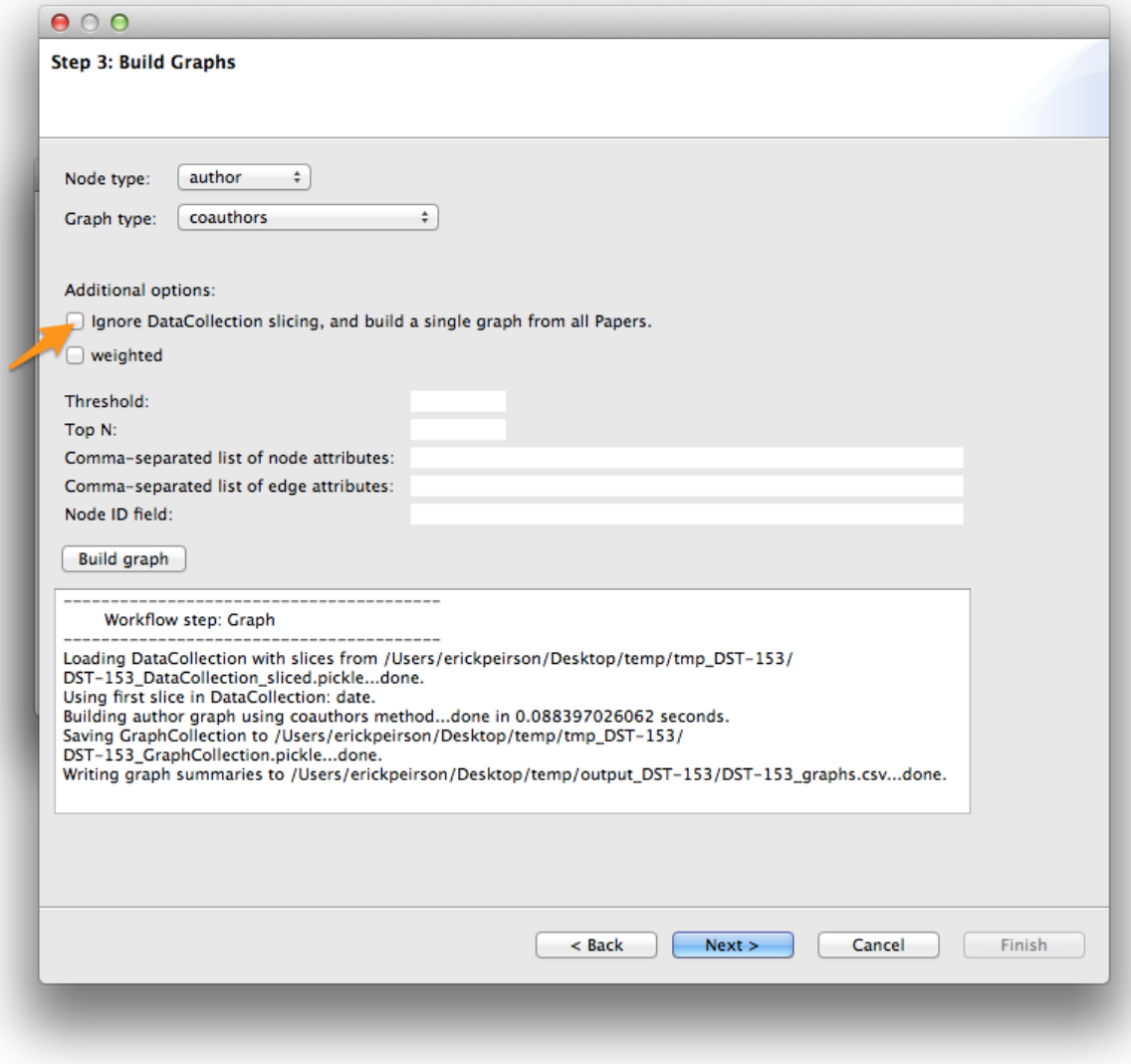
Use `--write-format xgmml` to select the dynamic XGMML export option.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --write --write-format xgmml
---------------------------------------
        Workflow step: Write
---------------------------------------
Loading GraphCollection from /tmp/exampleID_GraphCollection.pickle...done.
Writing graphs to /Users/erickpeirson/exampleOutput with format xgmml...done.
```

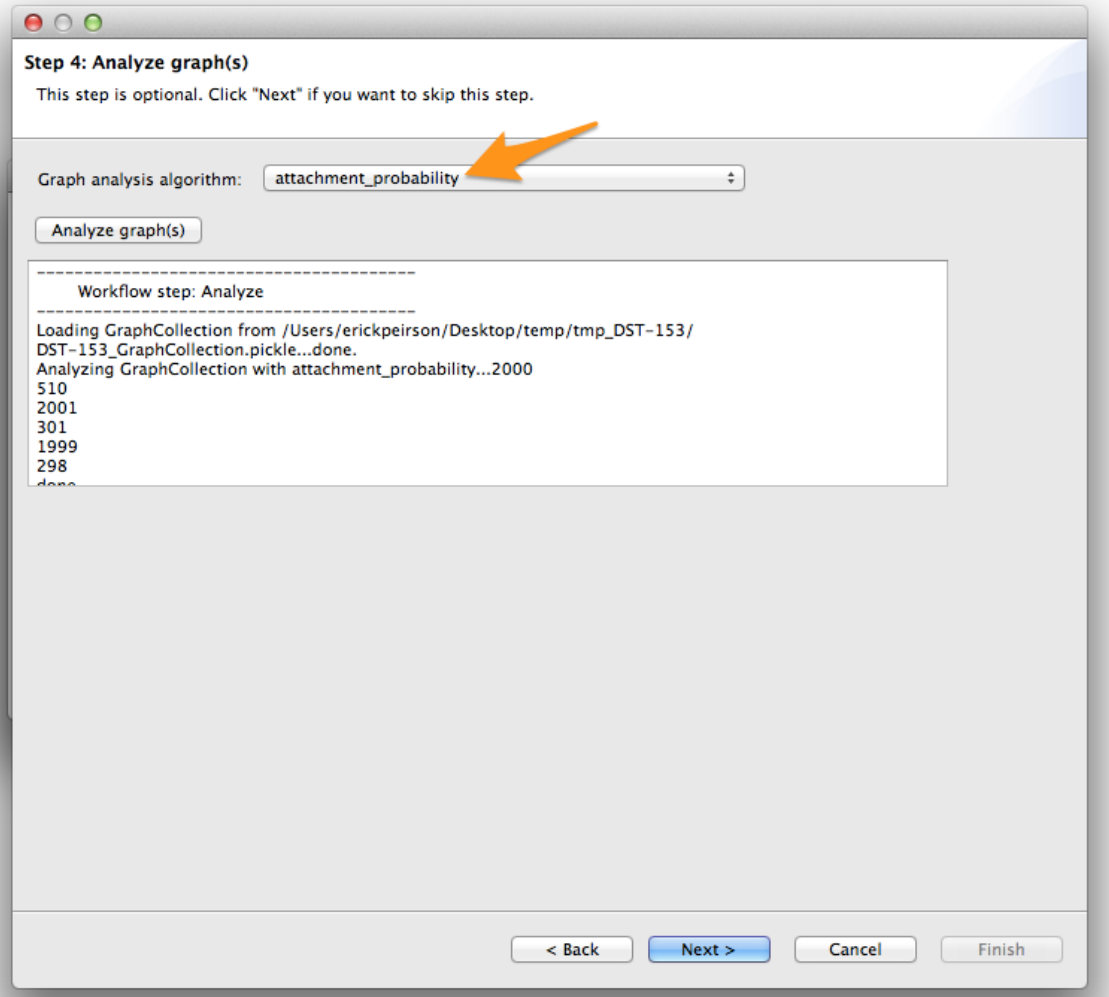
This should create a new file called `[DATASET_ID]_graph_dynamic.xgmml` in your output folder.

TethneGUI

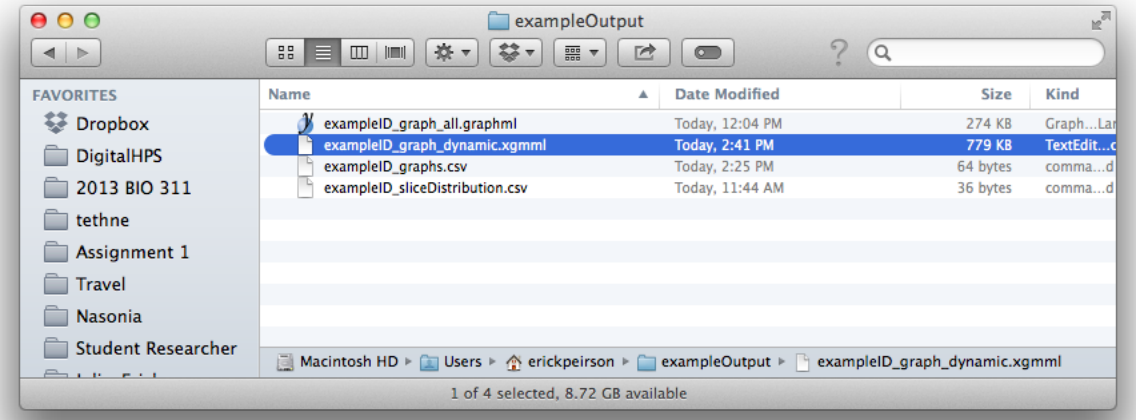
Use the `< Back` button to return to `Step 3: Build Graphs`. Uncheck the `Ignore DataCollection slicing` option, and then click the `Build graph` button again. Then click `Next >`.



At the `analysis` step, select `attachment_probability` from the `Graph analysis algorithm` menu, and click the `Analyze graph(s)` button. Then click `Next >`.



Finally, select `xgmml` in the `Output format` menu, and click `Write graph(s)`. This should create a new file called `[DATASET_ID]_graph_dynamic.xgmml` in your output folder.



Python

Use the `authorCollectionBuilder` to build a `GraphCollection` from your `DataCollection`.

```
>>> from tethne.builders import authorCollectionBuilder
>>> builder = authorCollectionBuilder(D)
>>> C = builder.build('date', 'coauthors')
```

The `analyze.collection.attachment_probability()` method automatically updates node attributes in your `GraphCollection`.

```
>>> import tethne.analyze as az
>>> az.collection.attachment_probability(C)
```

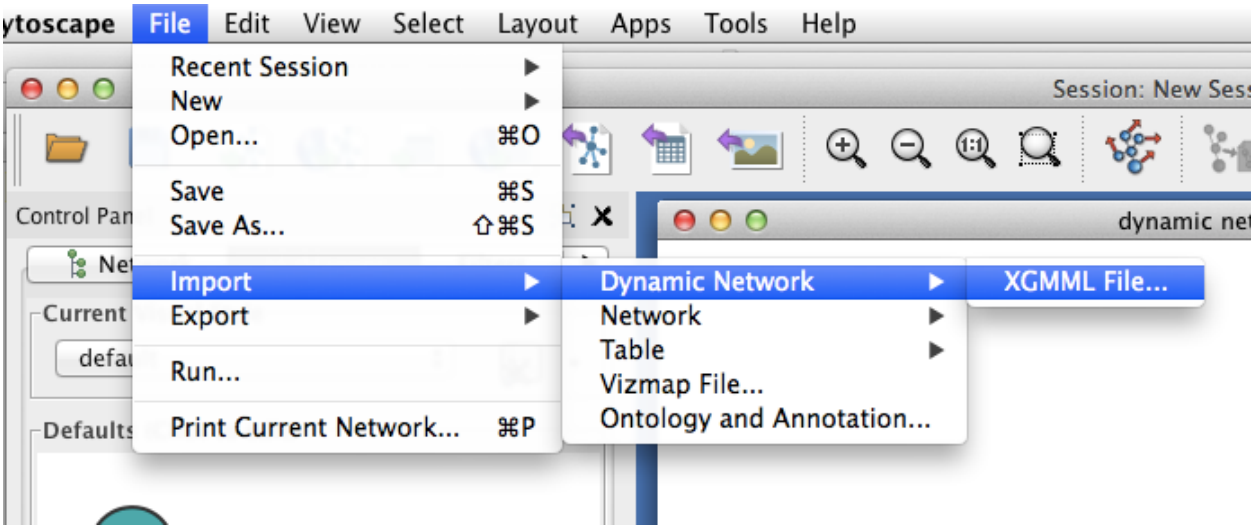
Use the `writers.collection.to_dxgmml()` method to create dynamic XGMML.

```
>>> import tethne.writers as wr
>>> wr.collection.to_dxgmml(C, "[OUTPUT_PATH]")
```

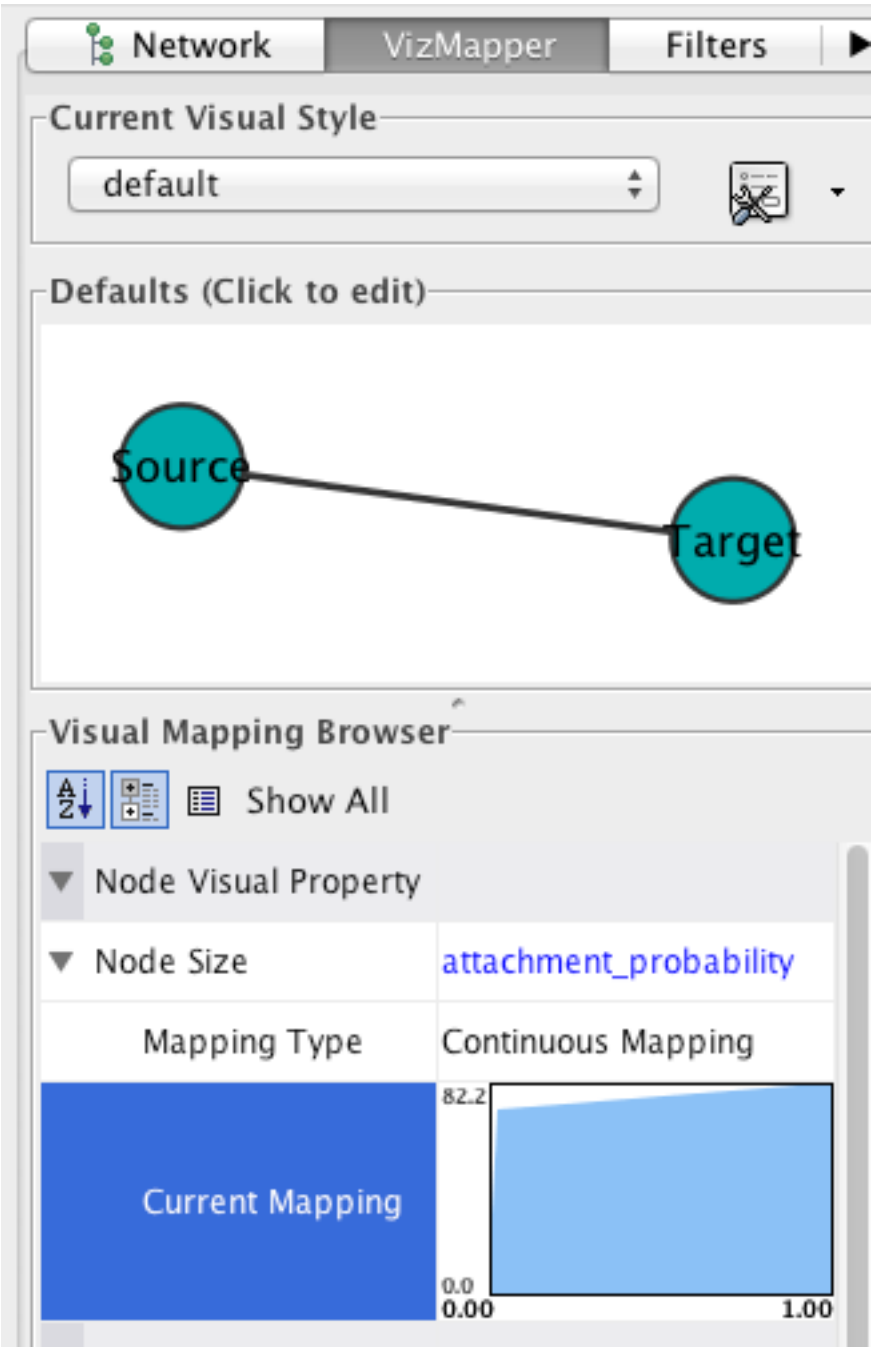
`[OUTPUT_PATH]` should be a path to the XGMML file that Tethne will create.

Visualizing a dynamic network in Cytoscape

In Cytoscape, import your .xgmml file by selecting `File > Import > Dynamic Network > XGMML File...`. Apply a force-directed or spring-embedded layout.

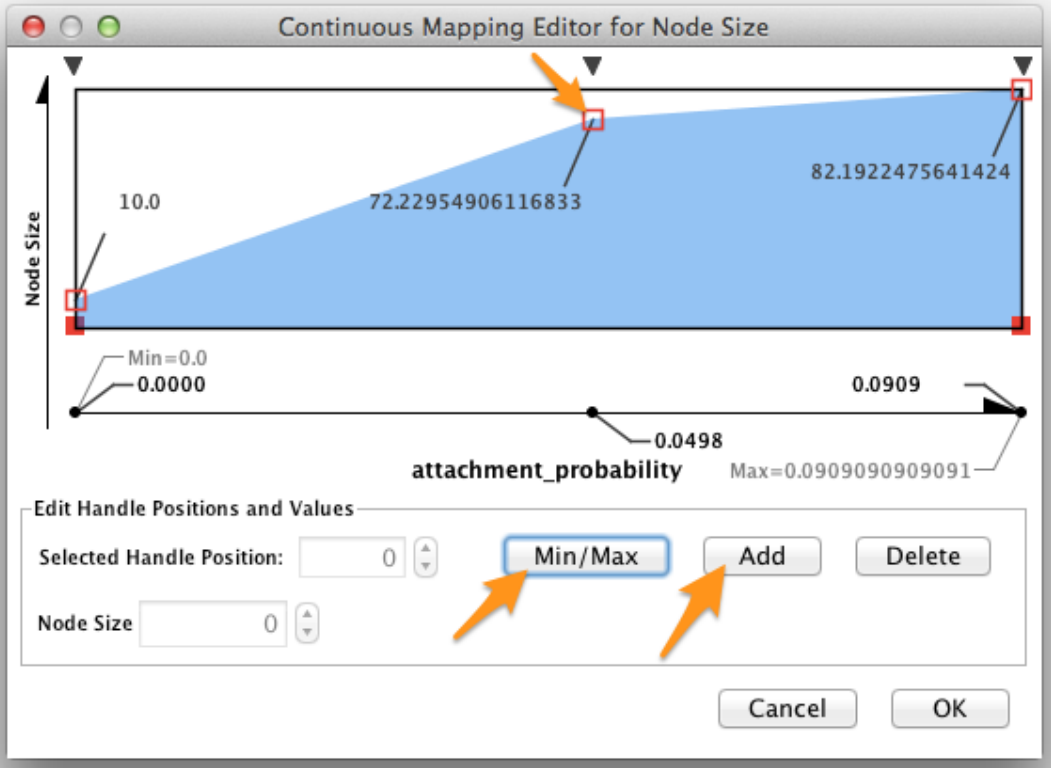


In the VizMapper, map `Node Size` to `attachment_probability`.



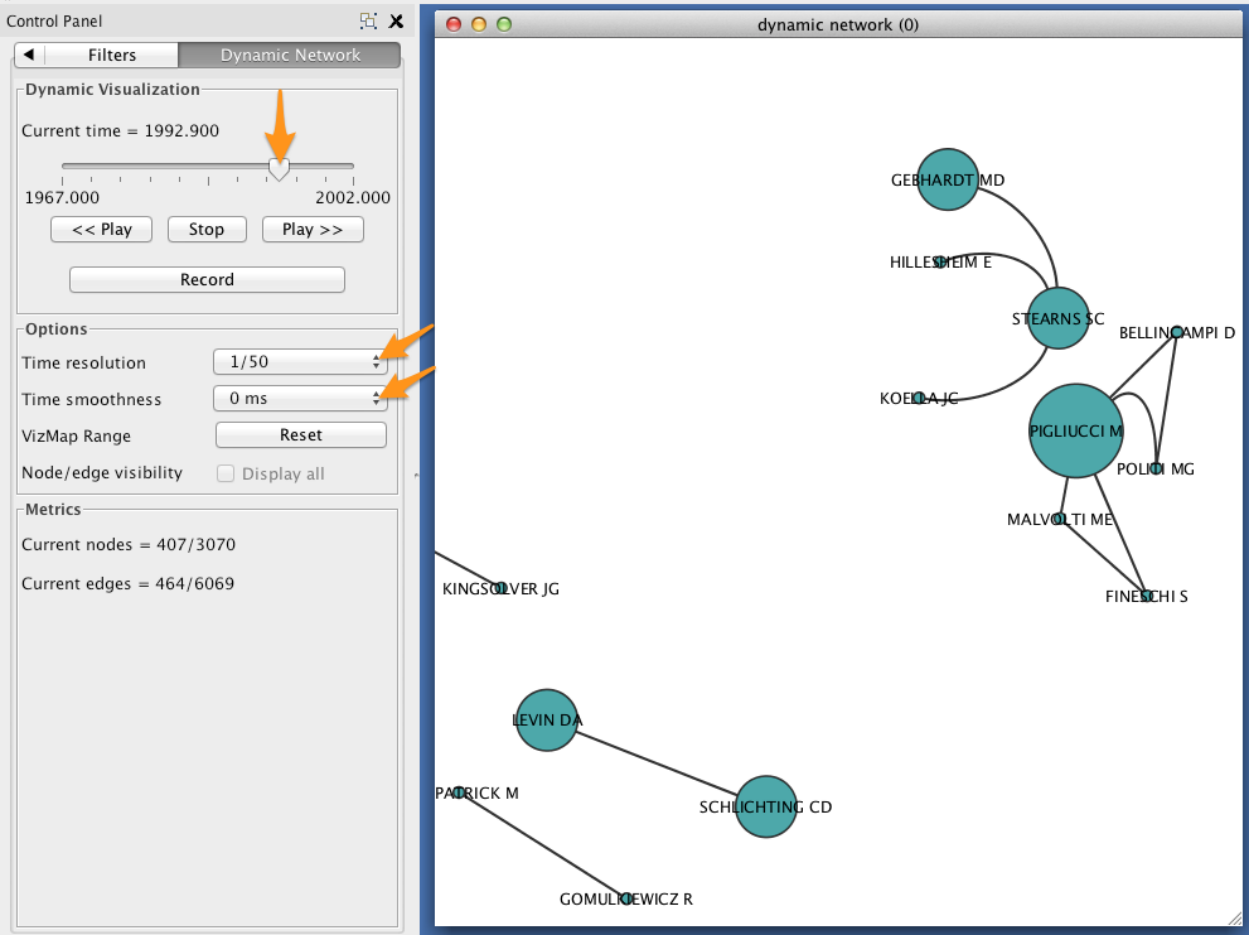
Double-click on the function icon next to `Current Mapping` to edit the `Node Size` mapping function.

1. Click the `Min/Max` button, and set the maximum value to `1.0`.
2. Slick on the `Add` button to create a new handle at an intermediate value. Drag the red open box up, and drag the corresponding black arrow left and right to alter the mapping function.
3. Click `OK`.

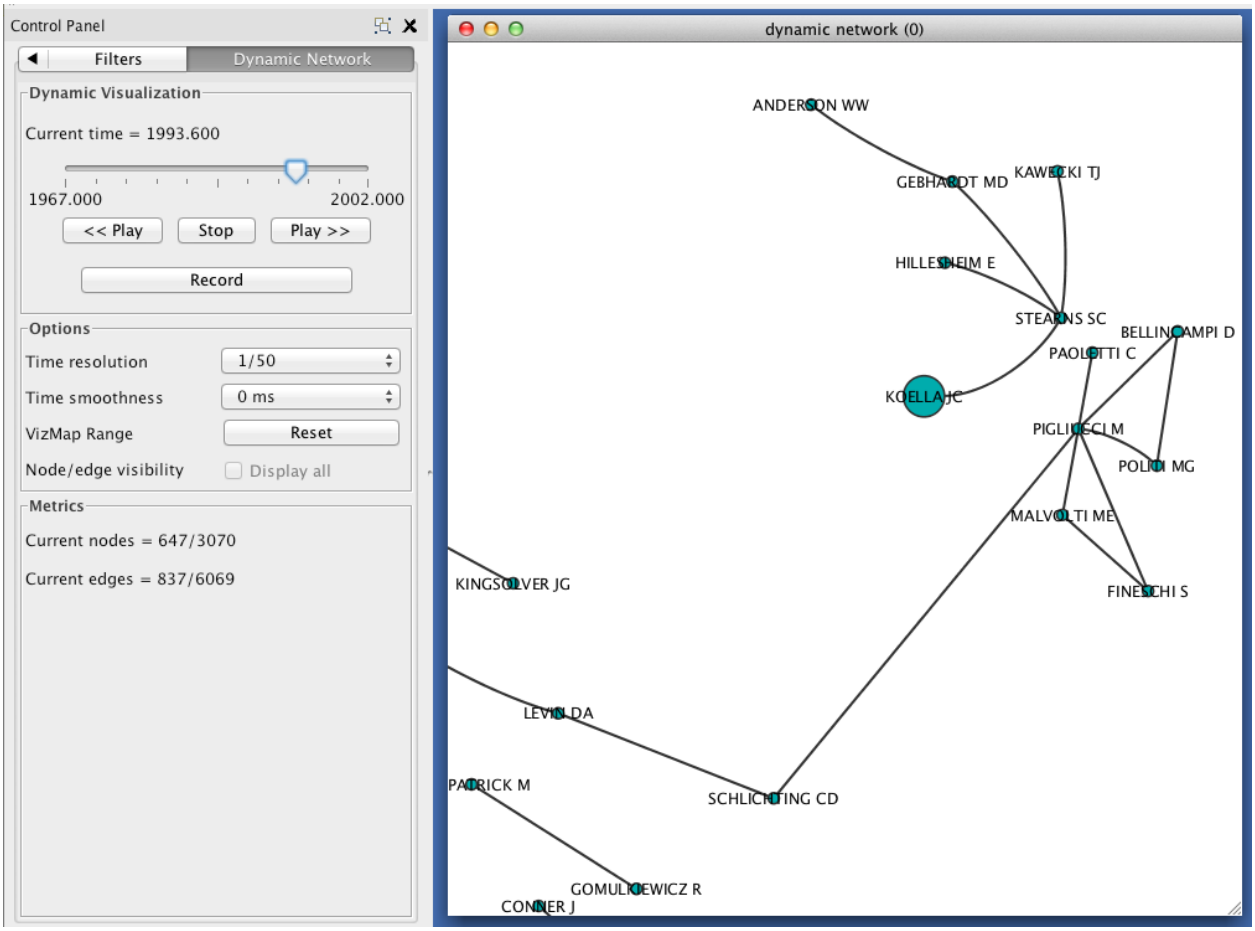


In the Control Panel, select the `Dynamic Network` tab.

1. Set the time resolution to roughly match the time-range of your network. In the example below, the network covers about 35 years, so a resolution of 1/50 was selected.
2. Set `Time smoothness` to `0 ms`.
3. Use the slider to move through the states of your dynamic network. To view all states in succession, use the `<< Play` and `Play >>` buttons.



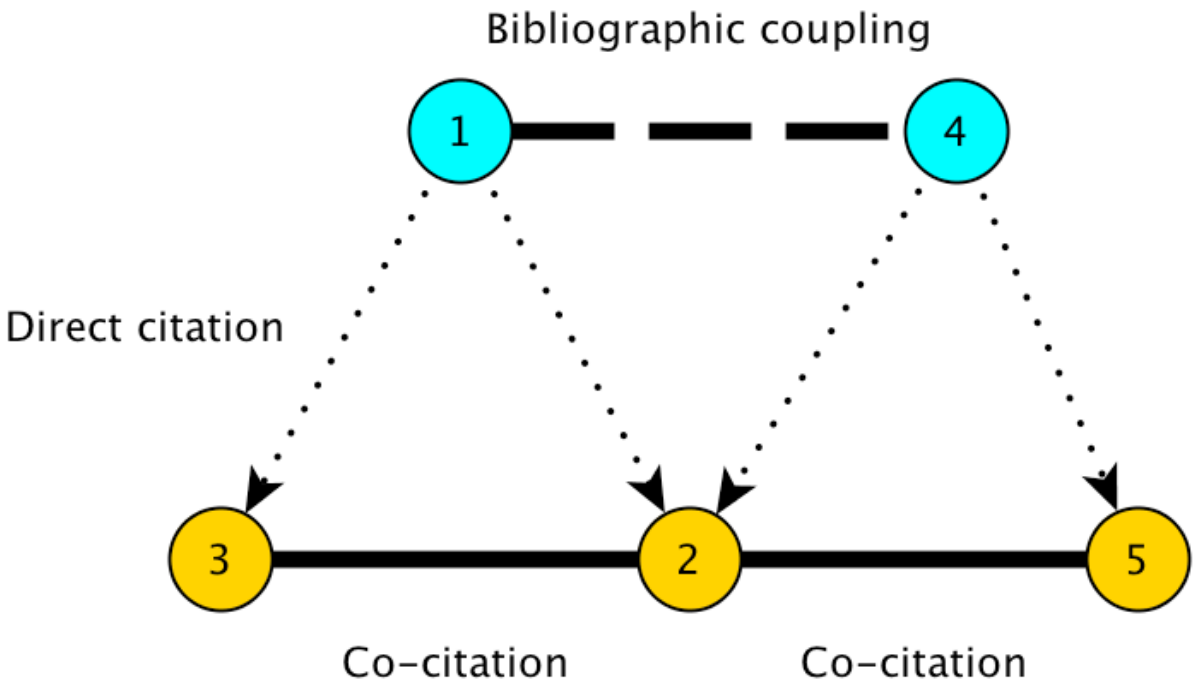
The size of each node should reflect the relative probability that a node will accrue a new neighbor in the next time slice. Try zooming in on a particular region of your network, and move between two successive states to verify that this is the case.



1.2.5 Bibliographic Coupling

Bibliographic coupling was first proposed as a method for detecting latent topical affinities among research publications by Myer M. Kessler at MIT in 1958. In 1972, J.C. Donohue suggested that bibliographic coupling could be used to the map "research fronts" in science, and this method, along with co-citation analysis and other citation-based clustering techniques, became a core methodology of the science-mapping craze of the 1970s. Bibliographic coupling is still employed in the context of both information-retrieval and science-studies.

Two papers are bibliographically coupled if they both cite at least some of the same papers. The core assumption of bibliographic coupling analysis is that if two papers cite similar literatures, then they must be topically related in some way. That is, they are more likely to be related to each other than to papers with which they share no cited references.



This tutorial provides a walk-through for building bibliographic coupling networks from Web of Science citation data, using the command-line interface, the TethneGUI (developed for demonstration purposes only), and the Python API.

The section *Cluster Detection* introduces Cytoscape's MCODE clustering app.

Before you begin, be sure to install the latest version of Tethne. Consult the *Installation* guide for details.

If you run into problems, don't panic. Tethne is under active development, and there are certainly bugs to be found. Please report any problems on our GitHub issue tracker.

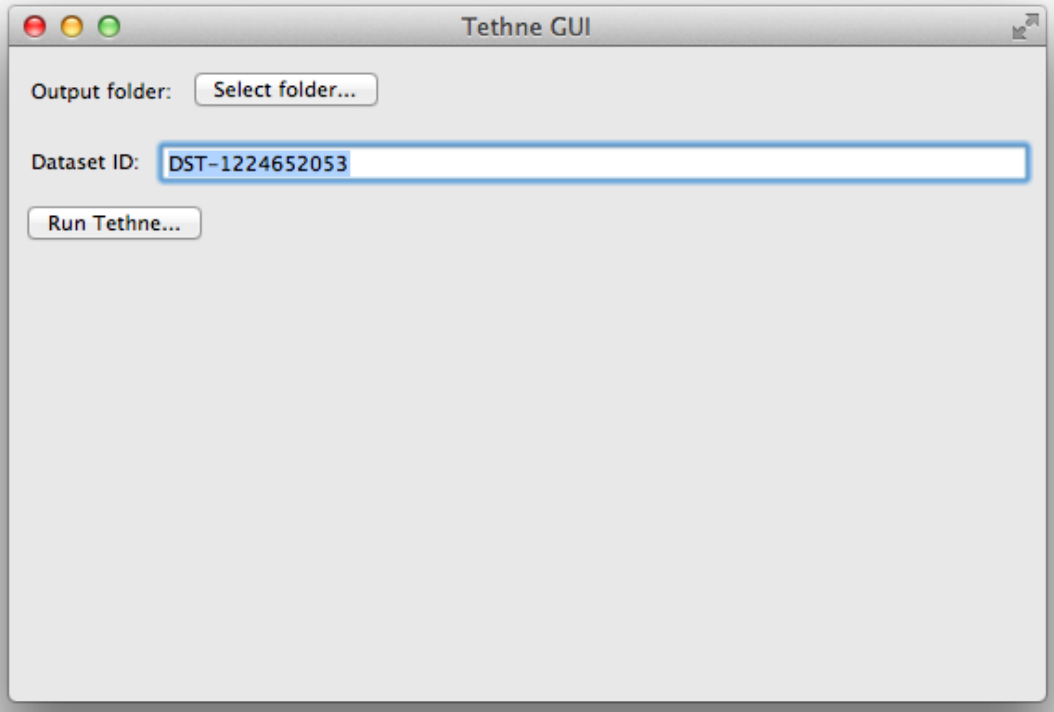
Getting Started

Before you start, you should choose an output folder where TethneGUI should store graphs and descriptions of your dataset.

You should also choose a dataset ID. This is a unique ID that Tethne will use to keep track of your data between workflow steps.

Initialize TethneGUI

When you first start TethneGUI, you should see a window like the one shown below. Click `Select folder...` to specify your output folder. A dataset ID should be automatically generated for you; you can change this if you wish.



Once you've selected an output folder and a dataset ID, click the `Run Tethne...` button.

Reading WoS Data

You can read WoS data from one or multiple field-tagged data files.

Command-line

Use `-I examplID` to specify your dataset ID, and `-O /Users/erickpeirson/exampleOutput` to specify your output folder.

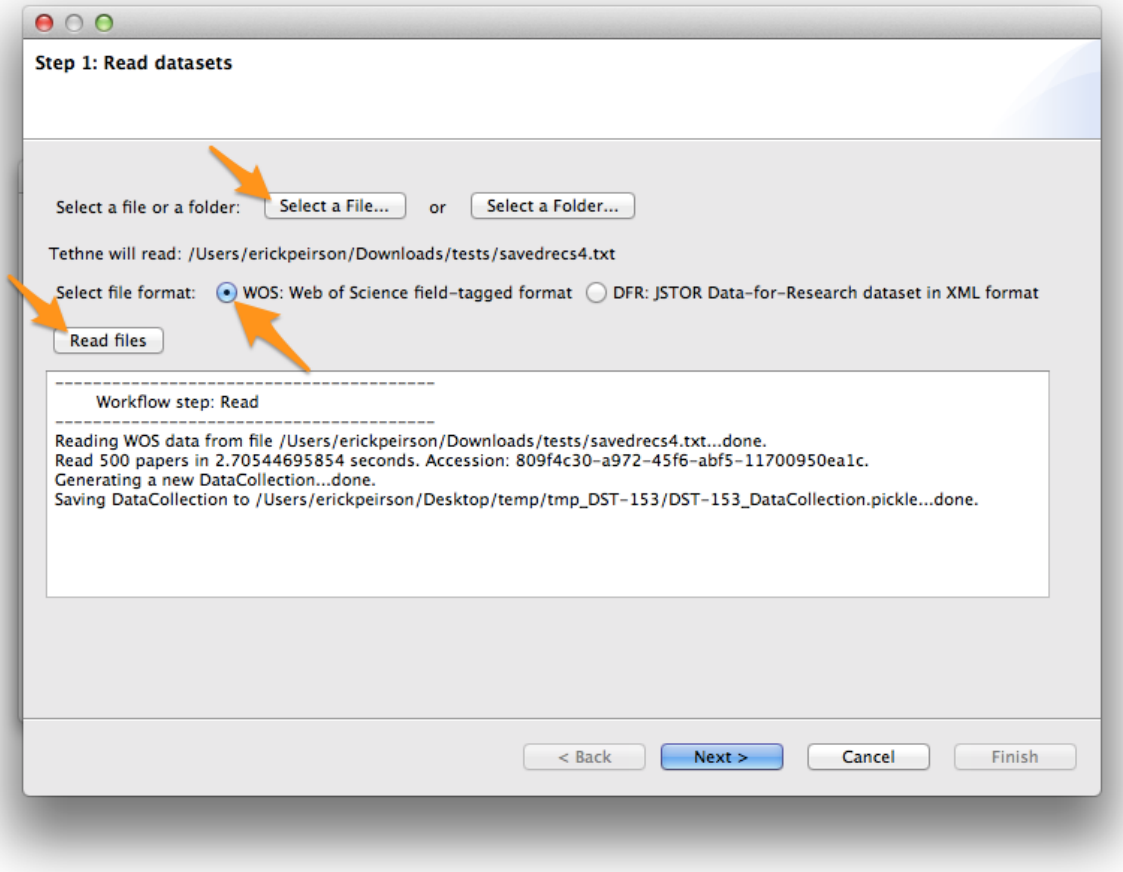
`--data-format=WOS` tells Tethne that your data are in the Web of Science field-tagged format.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --read-file \
--data-path=/Users/erickpeirson/Downloads/tests/savedrecs4.txt --data-format=WOS
----------------------------------------
        Workflow step: Read
----------------------------------------
Reading WOS data from file /Users/erickpeirson/Downloads/tests/savedrecs4.txt...done.
Read 500 papers in 2.67462515831 seconds. Accession: 0ff65dc3-b8f7-4bdc-a714-2d2a539f10a9.
Generating a new DataCollection...done.
Saving DataCollection to /tmp/exampleID_DataCollection.pickle...done.
```

TethneGUI

1. Select your WoS data file. If you have one data file, click the `Select a File...`. If you have multiple data files in their own folder, click `Select a Folder...`.
2. Select the `WOS` file format.
3. Click the `Read files` button.

Depending on the size of your dataset, this may take a minute or two. When TethneGUI is done reading your data, you should see messages like those depicted in the image below.



If your data are read successfully, click `Next >`.

Python

First import the `tethne.readers` module, then use the `readers.wos.read()` method to create a list of `Paper` instances. You can use `readers.wos.from_dir()` to import all of the WoS datafiles in a directory.

```
>>> # Parse data.
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/To/FirstDataSet.txt")
```

Then create a new `DataCollection` to organize your data.

```
>>> from tethne.data import DataCollection
>>> D = DataCollection(papers)
```

Slicing WoS Data

In this tutorial, we will first build a static bibliographic coupling network using all of the records in your WoS dataset. Then, if your dataset contains records from across a broad time-domain, you may also wish to view the evolution of your bibliographic coupling network over time by slicing your data using a `sliding time_window`. Since we can choose to merge our data slices in the `graph` step, we'll go ahead and slice our data now.

The sliding time-window slice method is a bit different than the simple time-period slice method used in the *Coauthorship Networks* tutorial. Whereas time-period slicing divides data into subsets by sequential non-overlapping time periods, subsets generated by time-window slicing can overlap.

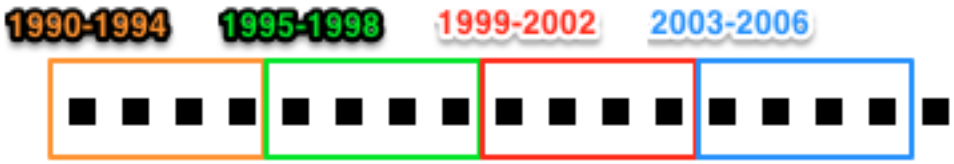


Figure 1.1: **Time-period** slicing, with a window-size of 4 years.

Figure 1.2: **Time-window** slicing, with a window-size of 4 years and a step-size of 1 year.

Command-line

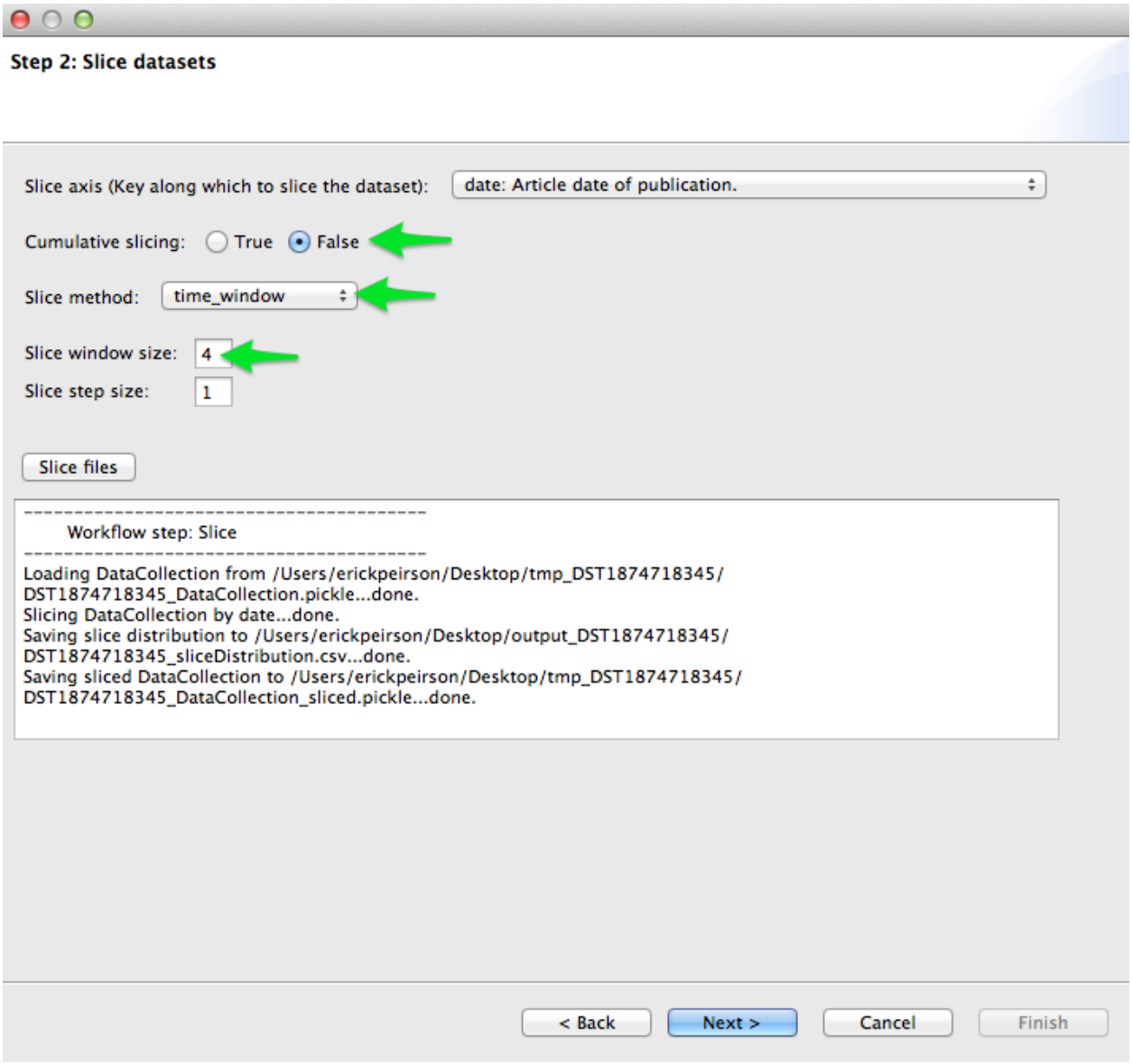
```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --slice -S date \
> -M time_window --window-size=4
----------------------------------------
        Workflow step: Slice
----------------------------------------
Loading DataCollection from /tmp/exampleID_DataCollection.pickle...done.
Slicing DataCollection by date...done.
Saving slice distribution to /Users/erickpeirson/exampleOutput/exampleID_sliceDistribution.csv...done
Saving sliced DataCollection to /tmp/exampleID_DataCollection_sliced.pickle...done.
```

TethneGUI

1. The slice axis should be set to `date` by default. If not, select it from the `Slice axis` drop-down menu.
2. Set `Cumulative slicing` to `False`.

3. Select `time_window` from the `Slice method` menu.
4. Set the `Slice window size` to `4`.
5. Click `Slice files`.

After a few minutes, slicing should be complete; click `Next >`.



Python

Use the `tethne.data.DataCollection.slice()` method to slice your data.

```
>>> D.slice('date', 'time_window', window_size=4)
```

Building the Bibliographic Coupling Graph

For now, we'll ignore data slicing and generate a single bibliographic coupling graph from the entire dataset using the `merged` option. Later on, we'll come back and use the data slicing to look at how the network evolves over time.

To generate a bibliographic coupling network, we will tell Tethne to use papers for nodes, and use the `bibliographic_coupling` graph type. For a complete list of graph types available in Tethne, see `networks`.

Generating an informative graph using bibliographic coupling will require some tuning. Depending on the criteria that you used to generate your bibliographic dataset, you may need to adjust the coupling `threshold`. Papers from a relatively narrow field have a high probability of sharing cited references, thus a threshold of `1` shared reference will result in a nearly complete graph that yields little information about the latent topical structure of that literature. If your dataset contains papers from quite disparate fields, however, you may wish to keep the threshold low.

Since papers vary widely in the total number of references that they cite, it may be desirable to use a normalized overlap value rather than an absolute one. If the `weighted` parameter is set to `True`, Tethne will use the normalized similarity metric `s`:

$$s = \frac{N_{i|j}}{\sqrt{N_i N_j}}$$

If you choose to use absolute overlap (`weighted` is `False`), we suggest starting with a `threshold` of `5`, and then adjusting it upward or downward to achieve optimal clustering. If you choose to use normalized overlap (`weighted` is `True`), then try starting with a `threshold` of `0.05`.

We'll also include some node attributes: `date`, `jtitle` (journal title), and `atitle` (article title).

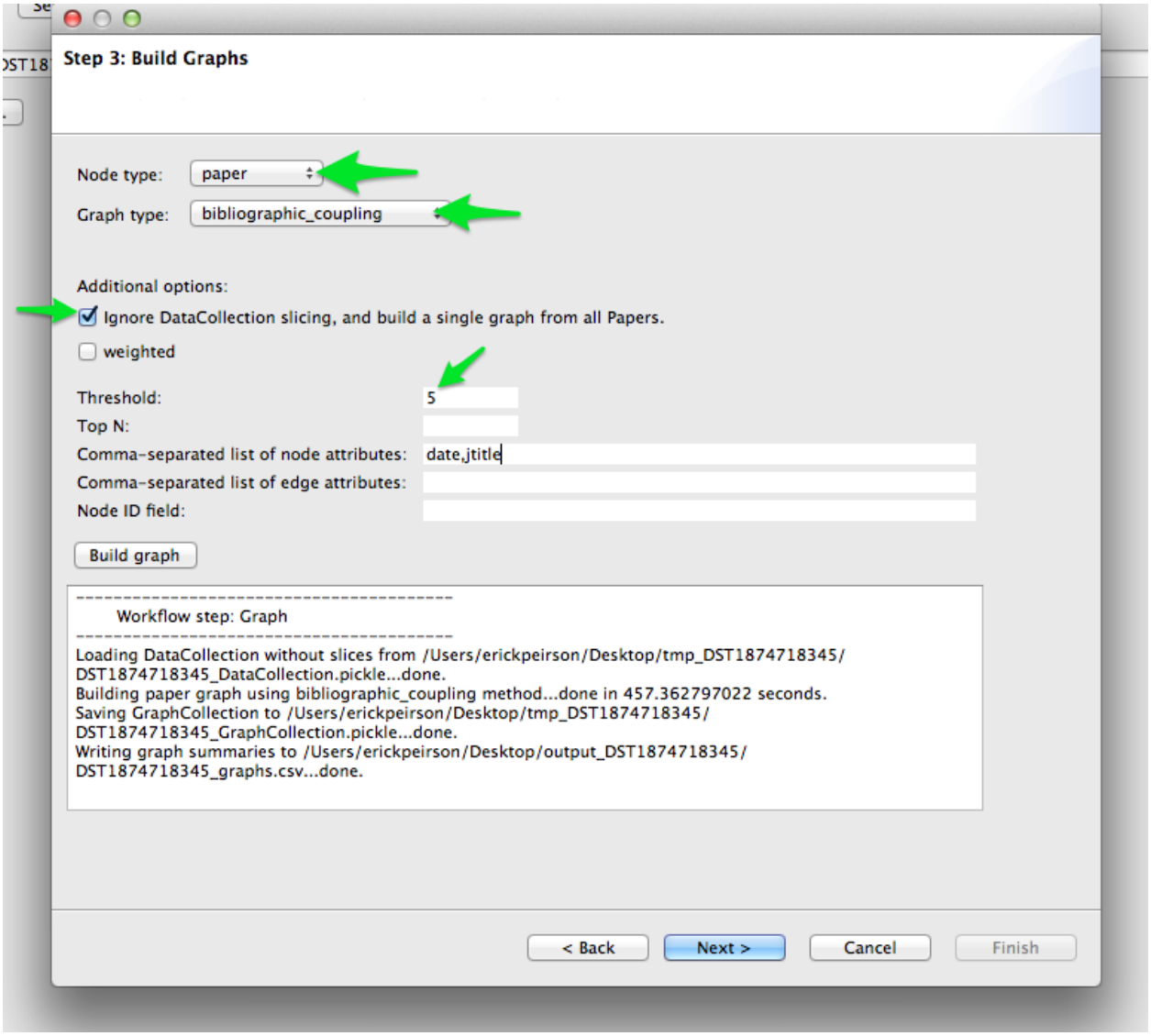
Command-line

The value of the `--node-attr` argument should be a list of keys from the `Paper` class, separated by commas (no spaces).

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --graph --merged \
> --node-type=paper --graph-type=bibliographic_coupling --threshold=0.05 --weighted \
> --node-attr=date,jtitle,atitle
----------------------------------------
        Workflow step: Graph
----------------------------------------
Loading DataCollection without slices from /tmp/exampleID_DataCollection.pickle...done.
Building author graph using coauthors method...done in 0.144234895706 seconds.
Saving GraphCollection to /tmp/exampleID_GraphCollection.pickle...done.
Writing graph summaries to /Users/erickpeirson/exampleOutput/exampleID_graphs.csv...done.
```

TethneGUI

Select `author` from the `Node type` menu, and `coauthors` from the `Graph type` menu. Check the `Ignore DataCollection slicing` option, then click `Build graph`.



Once the graph is built, click `Next >`. For now, we'll skip the analysis step. Click `Next >` again to reach `Step 5: Write graph(s).`

Python

To generate a single graph from your `DataCollection`, call the `coauthors()` method directly from the `networks.authors` module.

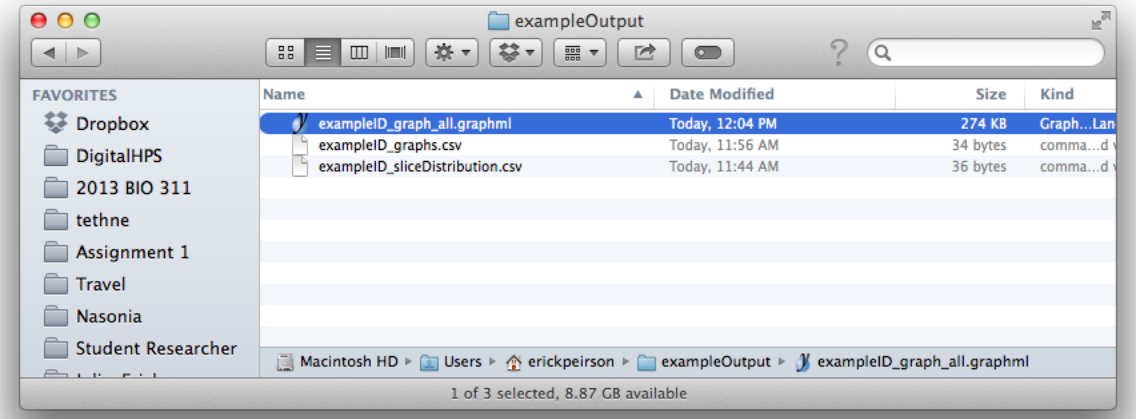
Use the `threshold` and `node_attribs` keyword arguments to set the minimum coupling threshold and node attributes, respectively. `node_attribs` should be a list of string keys from `Paper`.

```
>>> import tethne.networks as nt
>>> bc_graph = nt.papers.bibliographic_coupling(D.papers(), threshold=5,
...                                             node_attribs=['date','jtitle','atitle'])
```

Write the Graph to GraphML

GraphML is a widely-used static network data format. We will write our network to GraphML for visualization in Cytoscape.

This step should generate a file in your output folder called `[DATASET_ID]_graph_all.graphml`.

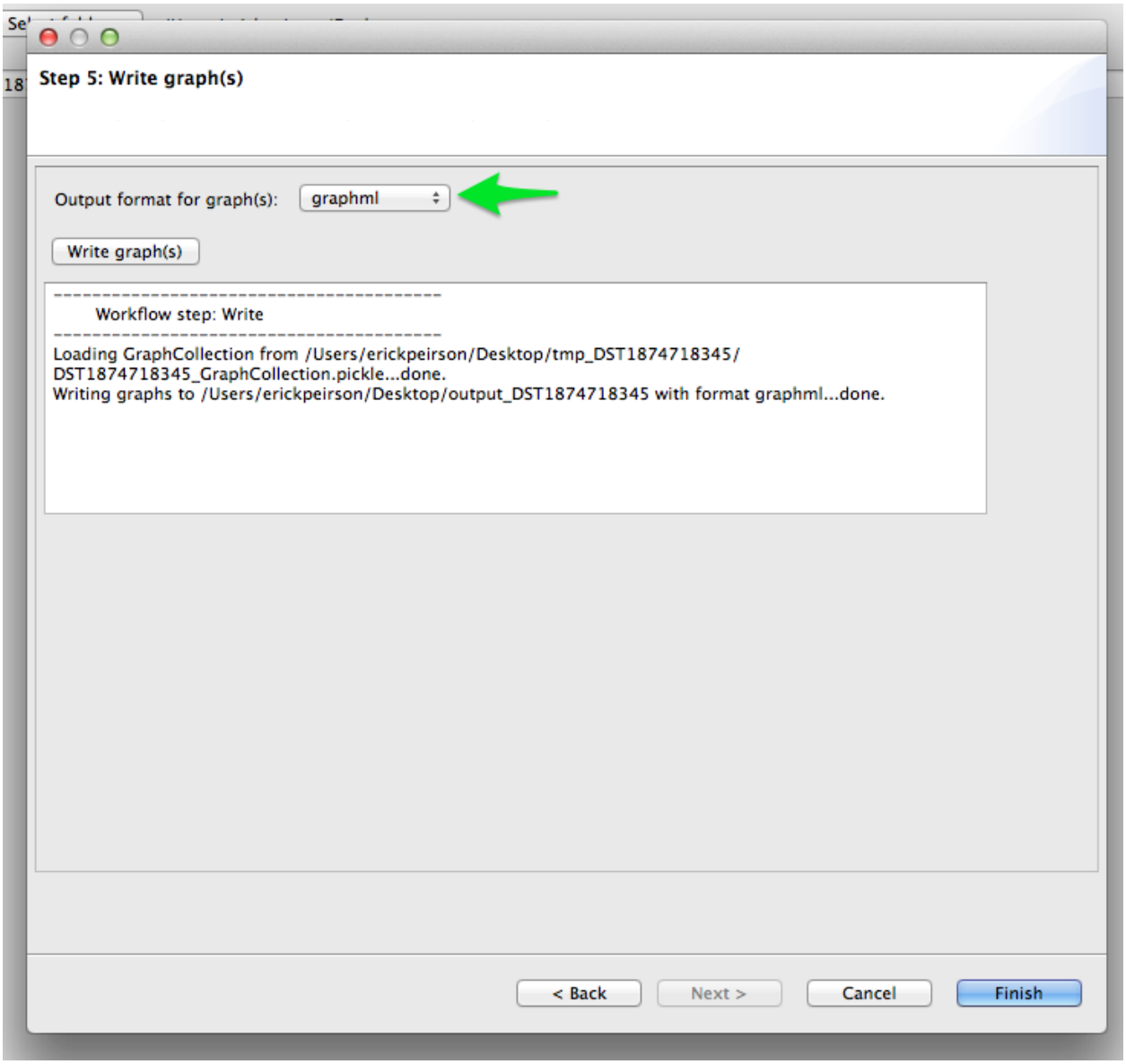


Command-line

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --write \
> --write-format graphml
----------------------------------------
        Workflow step: Write
----------------------------------------
Loading GraphCollection from /tmp/exampleID_GraphCollection.pickle...done.
Writing graphs to /Users/erickpeirson/exampleOutput with format graphml...done.
```

TethneGUI

Select `graphml` from the `Output format for graph(s)` menu, then click `Write graph(s)`.



Python

Use the `to_graphml()` method in `writers.collection` to create a GraphML data file.

```
>>> import tethne.writers as wr
>>> wr.graph.to_graphml(bc_graph, "[OUTPUT_PATH]")
```

`[OUTPUT_PATH]` should be a path to the GraphML file that Tethne will create.

Visualizing the Merged Network

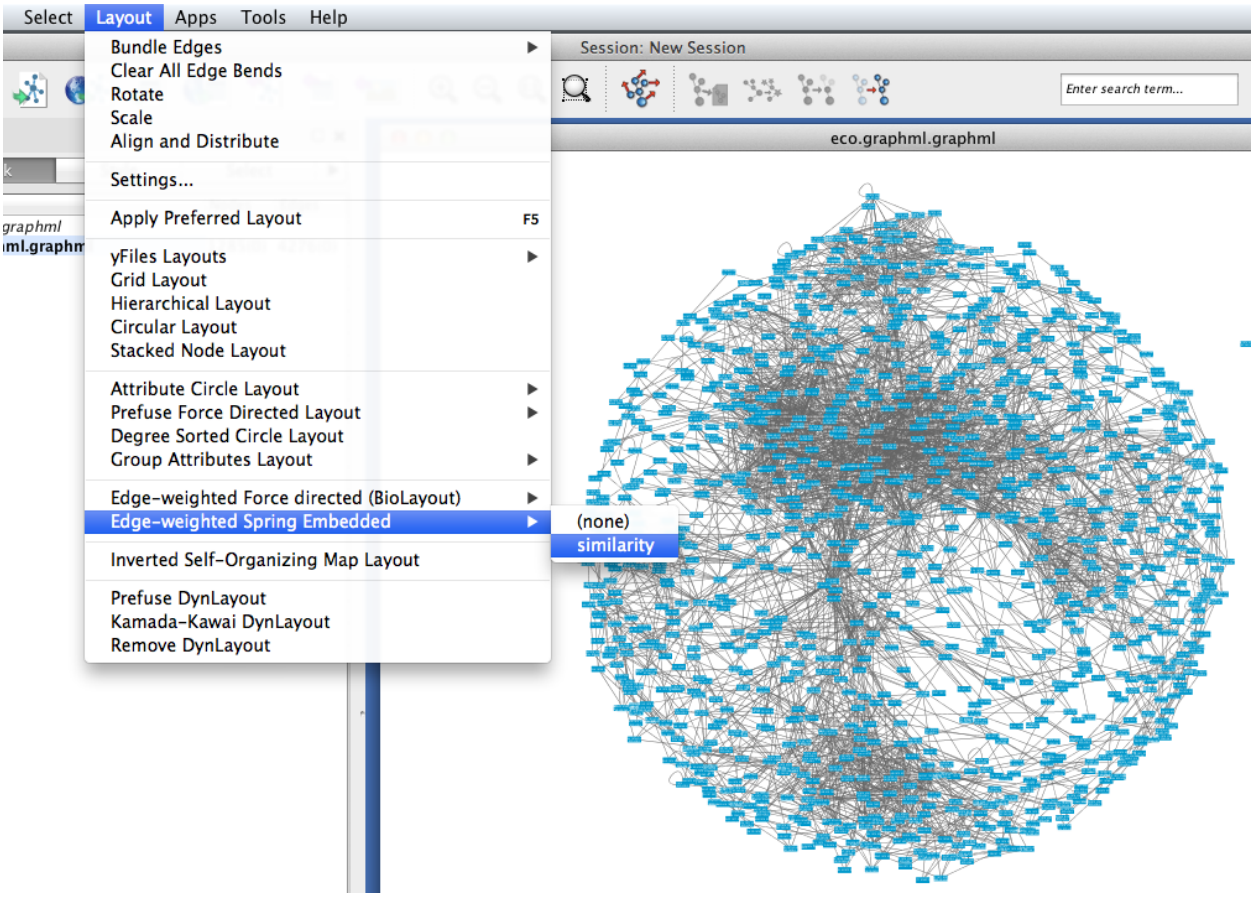
Cytoscape was developed in 2002, with funding from the National Instute of General Medical Sciences and the National Resource for Network Biology. The primary user base is the biomedical research community, especially systems biologists who study gene or protein interaction networks and pathways.

You can download Cytoscape 3 from url{http://www.cytoscape.org}. This tutorial assumes that you are using Cytoscape 3.1.

Import

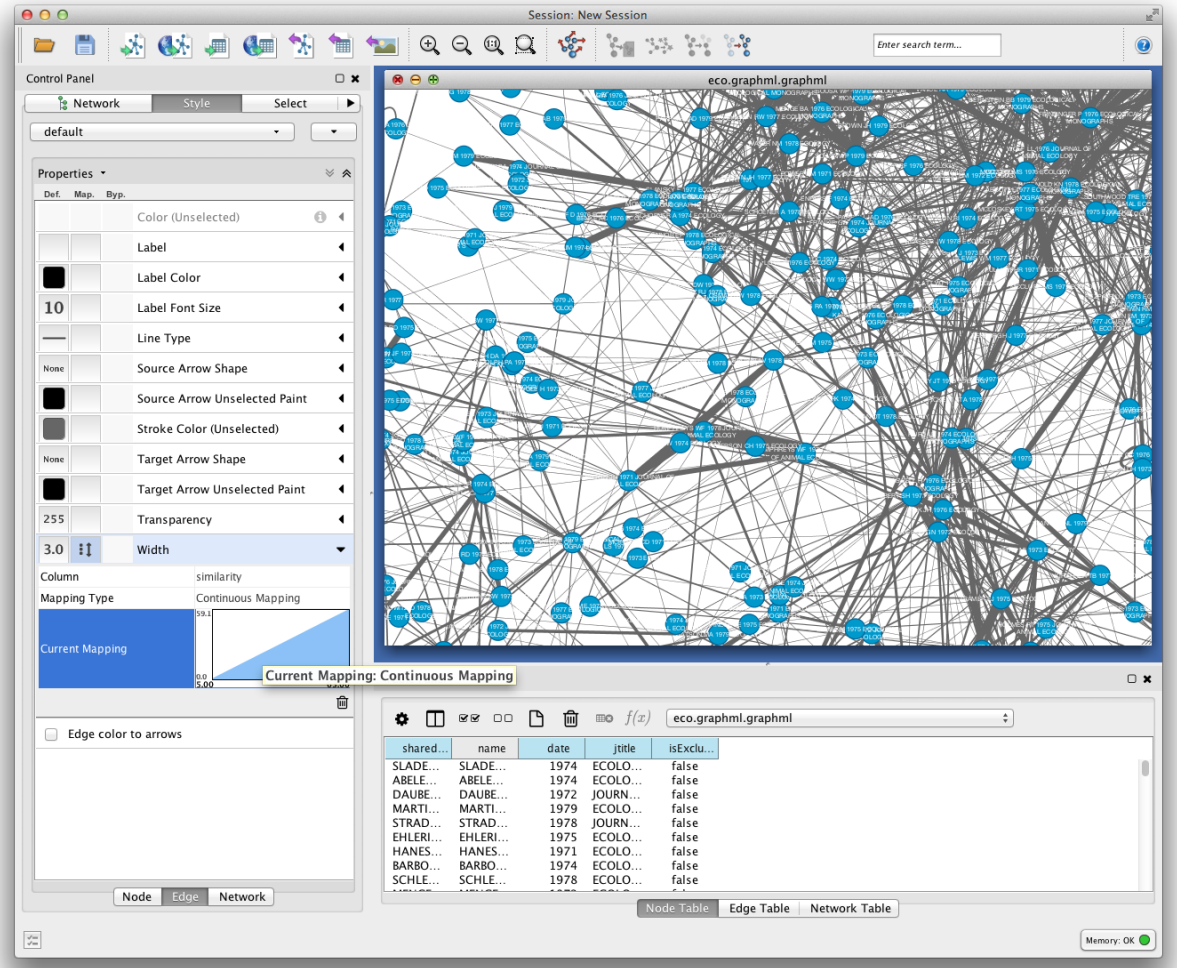
In Cytoscape, import your network by selecting `File > Import > Network > From file...` and selecting the GraphML file generated by Tethne in your output directory.

Tethne includes the `similarity` of each pair of papers as an edge attribute. You can tell Cytoscape to take similarity into account when laying out your graph. To apply an edge-weighted layout, select `Layout > Edge-weighted Spring Embedded > similarity`.

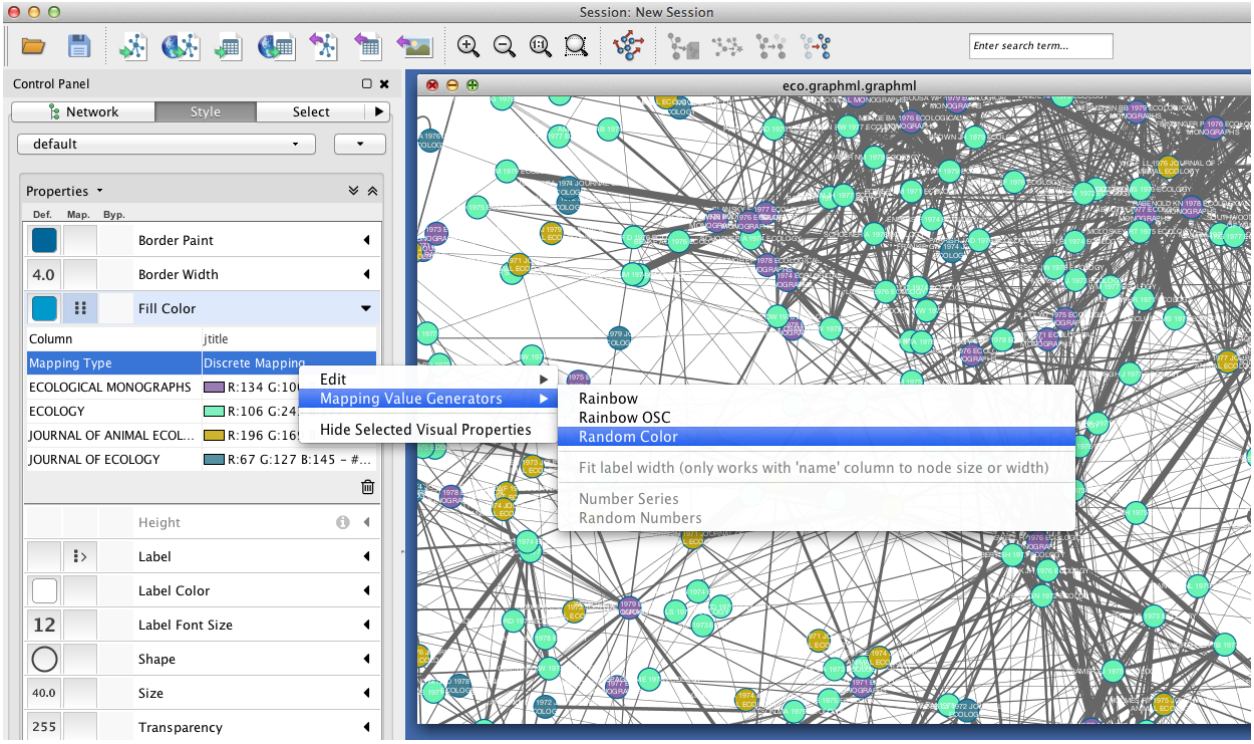


Your network may look like a giant hairball. If you can't see much structure at all, you may wish to go back and rebuild the graph with a higher threshold. If your network is very sparse, you may wish to lower the threshold.

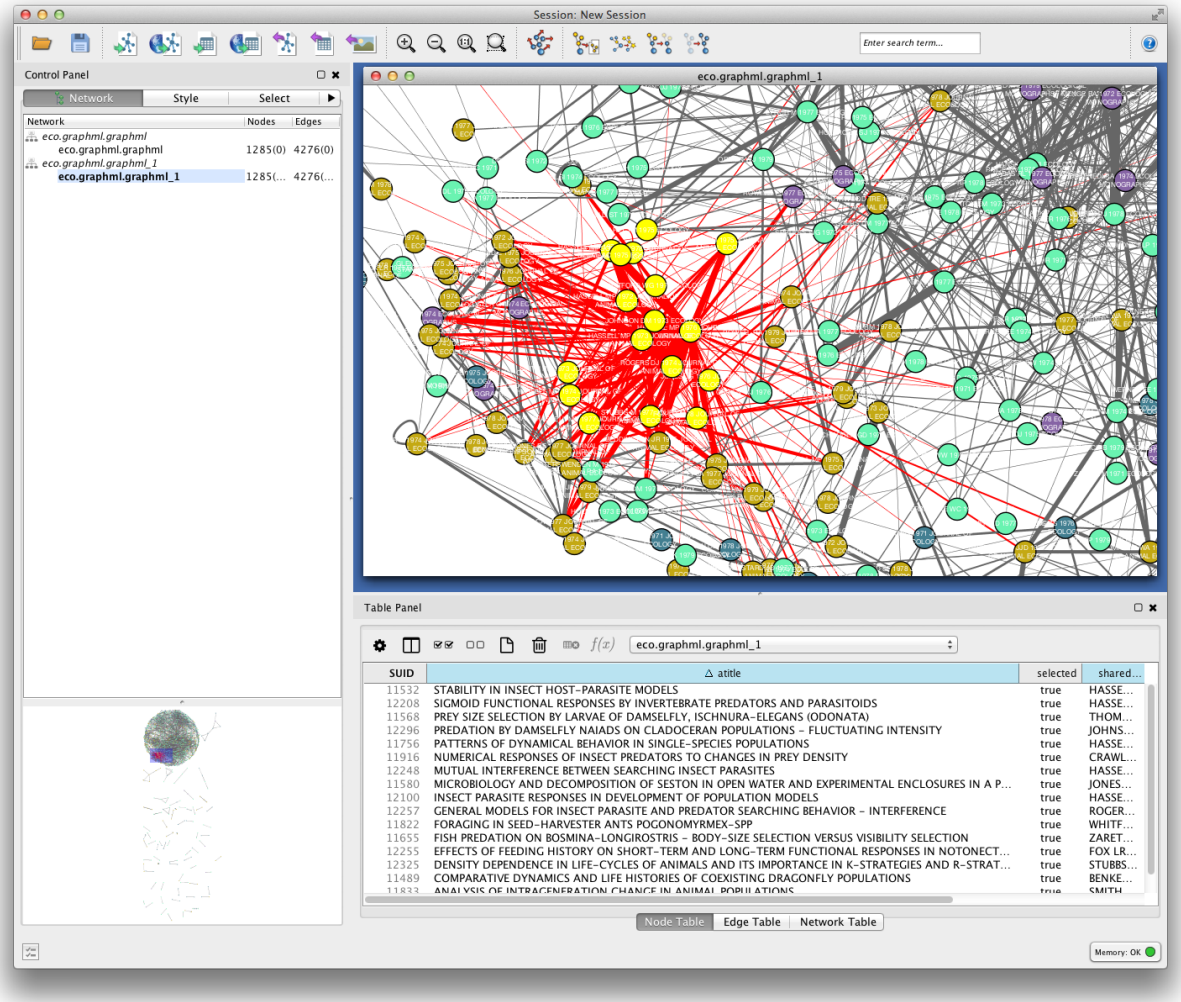
Set edge weight as a function of `similarity` to see which links are the strongest in your network.



To get some idea of whether certain clusters in the network correspond to publication in the same journal, set node fill color as a discrete function of `jtitle`. You can automatically generate node fill colors by right-clicking on the visual mapping, and selecting `Mapping Value Generators > Random Color`.



Since you included the title of each paper (`atitle`) as a node attribute, you can get some idea of what makes a particular region of the network hang together by selecting some nodes and inspecting the `Node Table` in the `Table Panel`. In the example below, a quick visual inspection suggests that parasites figure heavily in the selected papers.

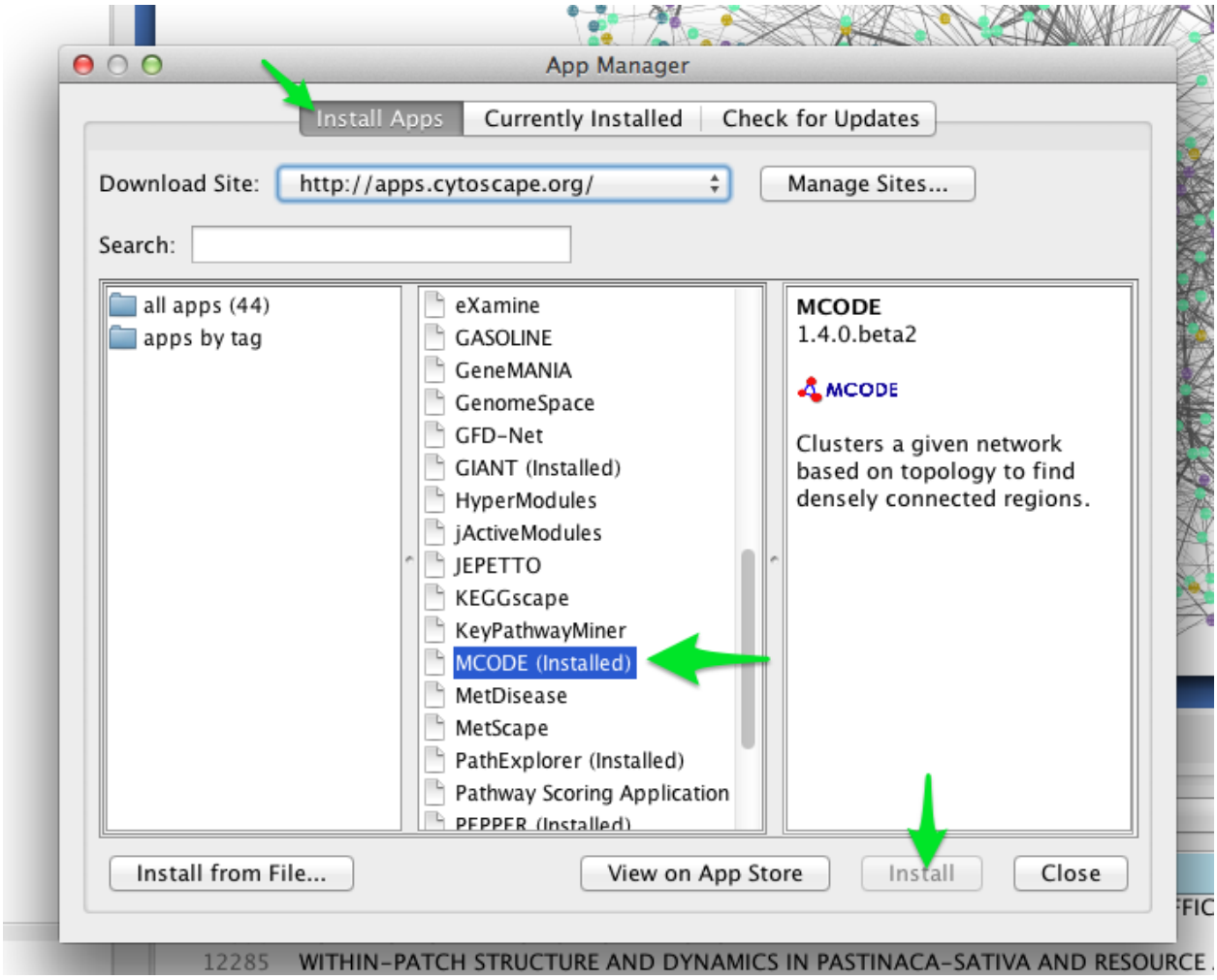


Cluster Detection

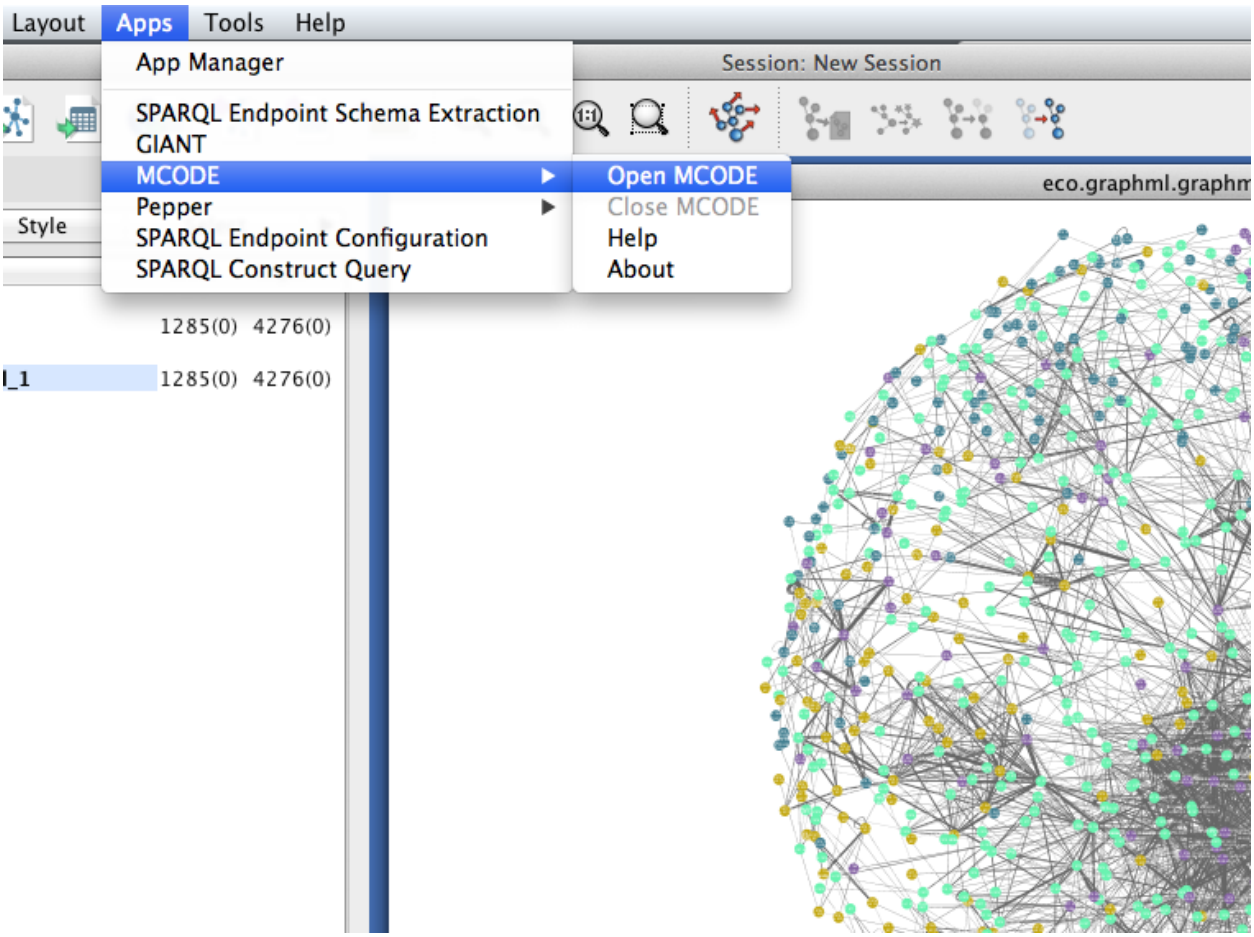
Especially if your network is very dense, it may be difficult to find salient clusters by visual inspection alone. Clustering algorithms provide a useful way to find groups of nodes that hang together in some way. Most clustering algorithms use an optimization function to find groups of nodes that are more densely connected among themselves than with the rest of the network.

One such clustering algorithm in Cytoscape is provided by the MCODE app. To install the MCODE app:

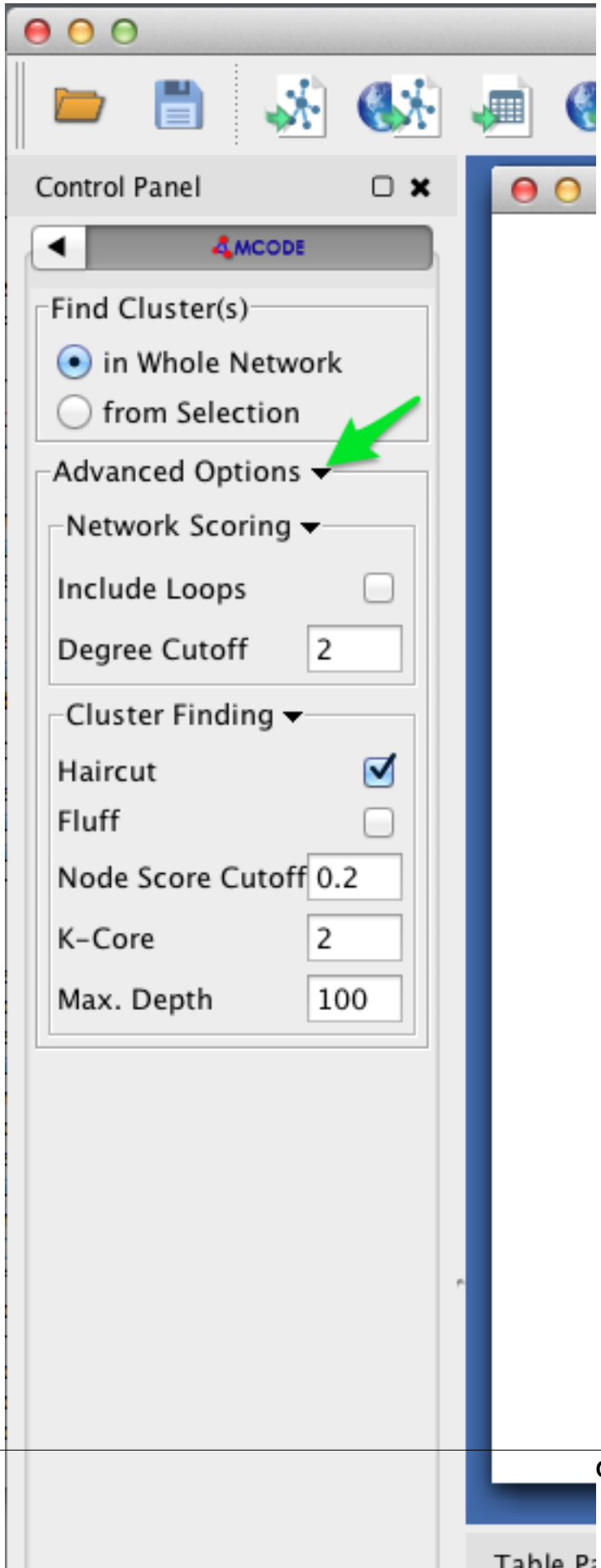
1. Select `Apps > App Manager` from the main menu.
2. Click on the `Install Apps` tab, and find MCODE in the list of available apps.
3. Click the `Install` button.



MCODE should now appear in the `Apps` menu.

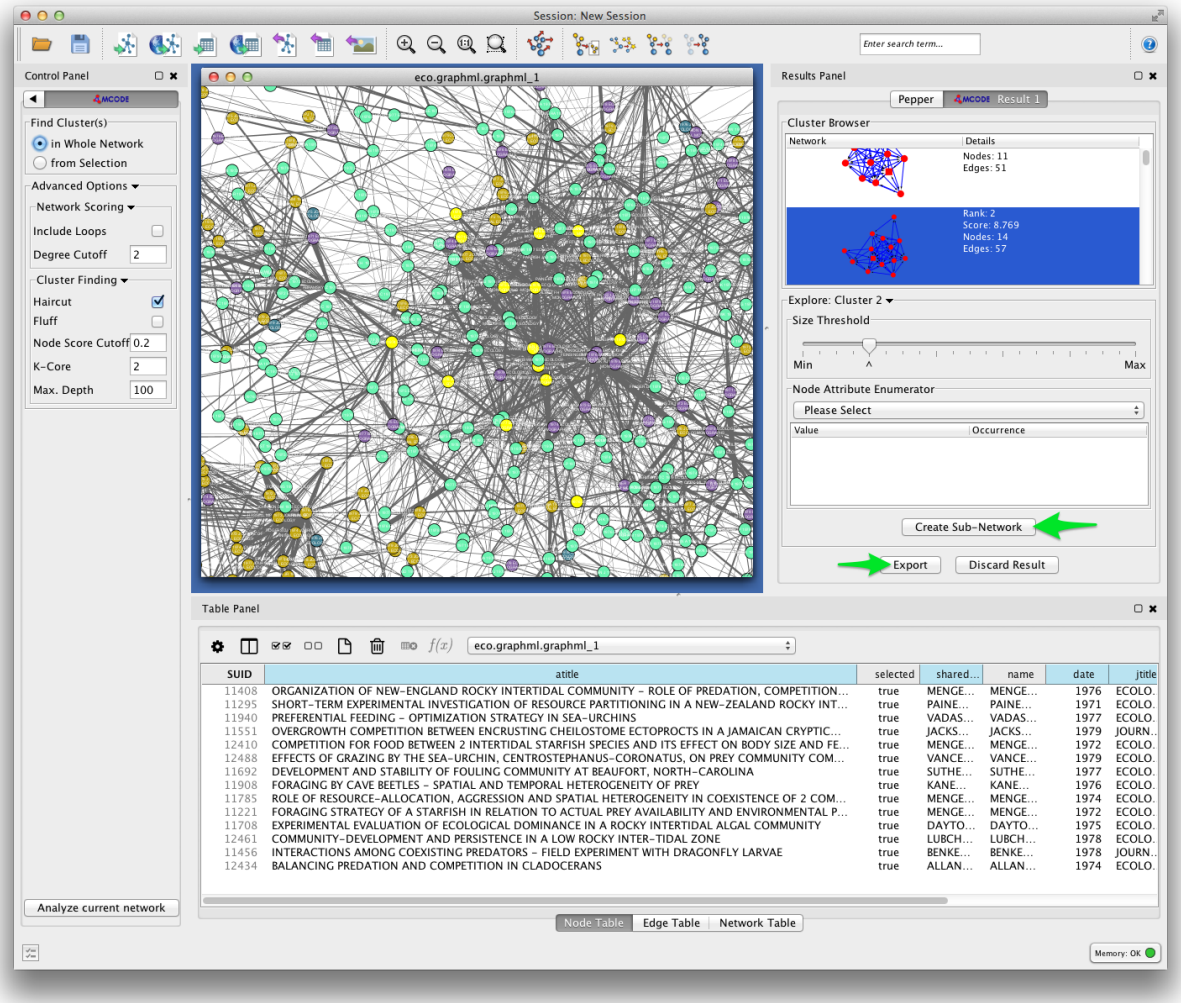


1. Select `Apps > MCODE > Open MCODE`. A new tab should appear in the `Control Panel` at left.
2. To adjust the parameters of the MCODE cluster-finding algorithm, expand the `Advanced Options`. MCODE works reasonable well with the default settings.
3. Click the `Analyze current network` button.


Control Panel
MCODE
Find Cluster(s)
in Whole Network
from Selection
Advanced Options
Network Scoring
Include Loops
Degree Cutoff
2
Cluster Finding
Haircut
Fluff
Node Score Cutoff
0.2
K-Core
2
Max. Depth
100

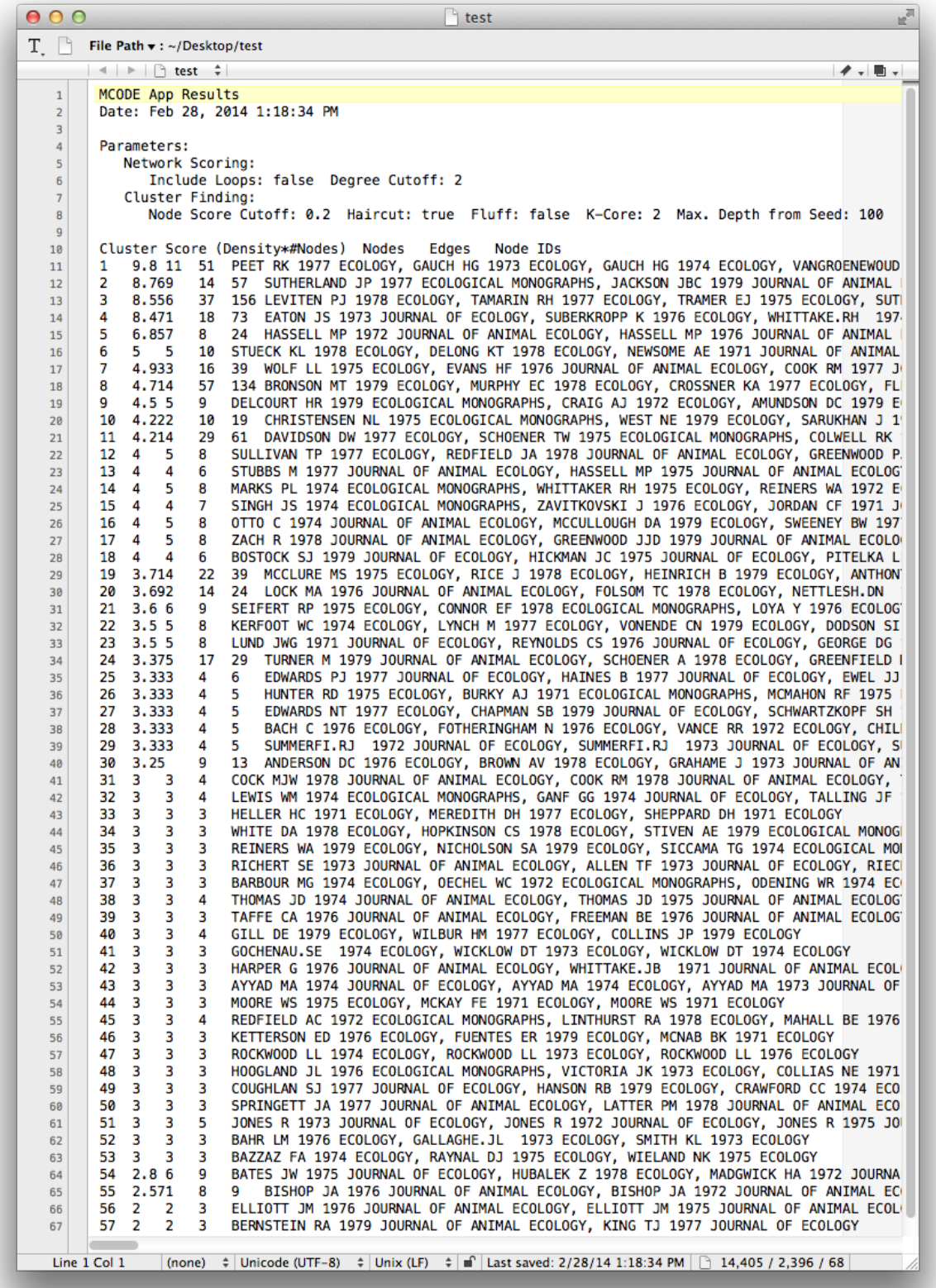
After a few moments, a new window should appear on the right side of the Cytoscape workspace. Click on a cluster in the `Cluster Browser` to select all of the nodes in that cluster. In some cases, MCODE will find clusters that are not at all obvious visually. This should give you an impression of the limitations of two-dimensional layouts for studying network structure, especially in very large, dense networks.

In the example below, MCODE has found a cluster of papers dealing with invertebrate predators in marine inter-tidal zones.



MCODE allows you to create a subnetwork from the selected cluster, or export your results. Exporting your results produces a table like the one shown below, listing each of the detected clusters and the papers the belong to them.

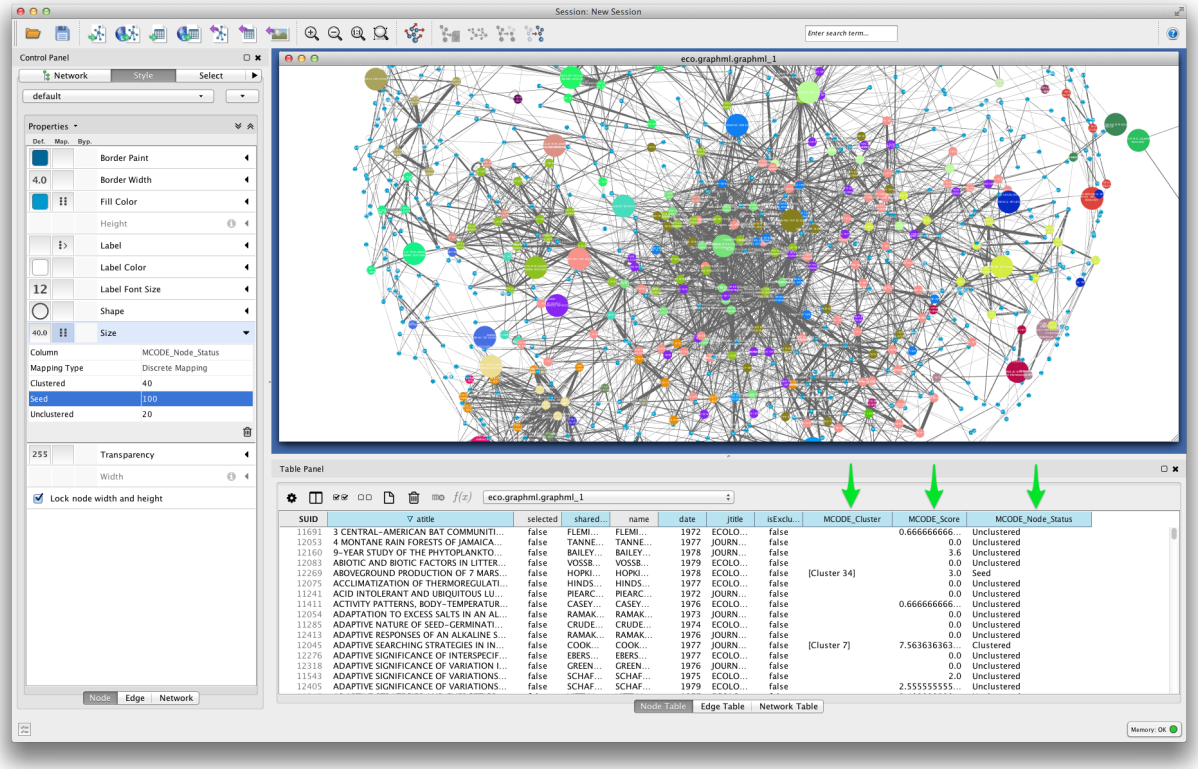
Future versions of Tethne will use this result to generate labels for each cluster based on the terms that uniquely characterize those groups of papers.


test
File Path : ~/Desktop/test
test
MCODE App Results
Date: Feb 28, 2014 1:18:34 PM
Parameters:
Network Scoring:
Include Loops: false Degree Cutoff: 2
Cluster Finding:
Node Score Cutoff: 0.2 Haircut: true Fluff: false K-Core: 2 Max. Depth from Seed: 100
Cluster Score (Density*#Nodes) Nodes Edges Node IDs
1 9.8 11 51 PEET RK 1977 ECOLOGY, GAUCH HG 1973 ECOLOGY, GAUCH HG 1974 ECOLOGY, VANGROENEWOUD
2 8.769 14 57 SUTHERLAND JP 1977 ECOLOGICAL MONOGRAPHS, JACKSON JBC 1979 JOURNAL OF ANIMAL
3 8.556 37 156 LEVITEN PJ 1978 ECOLOGY, TAMARIN RH 1977 ECOLOGY, TRAMER EJ 1975 ECOLOGY, SUT
4 8.471 18 73 EATON JS 1973 JOURNAL OF ECOLOGY, SUBERKROPP K 1976 ECOLOGY, WHITTAKE.RH 197
5 6.857 8 24 HASSELL MP 1972 JOURNAL OF ANIMAL ECOLOGY, HASSELL MP 1976 JOURNAL OF ANIMAL
6 5 5 10 STUECK KL 1978 ECOLOGY, DELONG KT 1978 ECOLOGY, NEWSOME AE 1971 JOURNAL OF ANIMAL
7 4.933 16 39 WOLF LL 1975 ECOLOGY, EVANS HF 1976 JOURNAL OF ANIMAL ECOLOGY, COOK RM 1977 J
8 4.714 57 134 BRONSON MT 1979 ECOLOGY, MURPHY EC 1978 ECOLOGY, CROSSNER KA 1977 ECOLOGY, FL
9 4.5 5 9 DELCOURT HR 1979 ECOLOGICAL MONOGRAPHS, CRAIG AJ 1972 ECOLOGY, AMUNDSON DC 1979 E
10 4.222 10 19 CHRISTENSEN NL 1975 ECOLOGICAL MONOGRAPHS, WEST NE 1979 ECOLOGY, SARUKHAN J 1
11 4.214 29 61 DAVIDSON DW 1977 ECOLOGY, SCHOENER TW 1975 ECOLOGICAL MONOGRAPHS, COLWELL RK
12 4 5 8 SULLIVAN TP 1977 ECOLOGY, REDFIELD JA 1978 JOURNAL OF ANIMAL ECOLOGY, GREENWOOD P
13 4 4 6 STUBBS M 1977 JOURNAL OF ANIMAL ECOLOGY, HASSELL MP 1975 JOURNAL OF ANIMAL ECOLOG
14 4 5 8 MARKS PL 1974 ECOLOGICAL MONOGRAPHS, WHITTAKER RH 1975 ECOLOGY, REINERS WA 1972 E
15 4 4 7 SINGH JS 1974 ECOLOGICAL MONOGRAPHS, ZAVITKOVSKI J 1976 ECOLOGY, JORDAN CF 1971 J
16 4 5 8 OTTO C 1974 JOURNAL OF ANIMAL ECOLOGY, MCCULLOUGH DA 1979 ECOLOGY, SWEENEY BW 197
17 4 5 8 ZACH R 1978 JOURNAL OF ANIMAL ECOLOGY, GREENWOOD JJD 1979 JOURNAL OF ANIMAL ECOLO
18 4 4 6 BOSTOCK SJ 1979 JOURNAL OF ECOLOGY, HICKMAN JC 1975 JOURNAL OF ECOLOGY, PITELKA L
19 3.714 22 39 MCCLURE MS 1975 ECOLOGY, RICE J 1978 ECOLOGY, HEINRICH B 1979 ECOLOGY, ANTHON
20 3.692 14 24 LOCK MA 1976 JOURNAL OF ANIMAL ECOLOGY, FOLSOM TC 1978 ECOLOGY, NETTLESH.DN
21 3.6 6 9 SEIFERT RP 1975 ECOLOGY, CONNOR EF 1978 ECOLOGICAL MONOGRAPHS, LOYA Y 1976 ECOLOG
22 3.5 5 8 KERFOOT WC 1974 ECOLOGY, LYNCH M 1977 ECOLOGY, VONENDE CN 1979 ECOLOGY, DODSON SI
23 3.5 5 8 LUND JWG 1971 JOURNAL OF ECOLOGY, REYNOLDS CS 1976 JOURNAL OF ECOLOGY, GEORGE DG
24 3.375 17 29 TURNER M 1979 JOURNAL OF ANIMAL ECOLOGY, SCHOENER A 1978 ECOLOGY, GREENFIELD
25 3.333 4 6 EDWARDS PJ 1977 JOURNAL OF ECOLOGY, HAINES B 1977 JOURNAL OF ECOLOGY, EWEL JJ
26 3.333 4 5 HUNTER RD 1975 ECOLOGY, BURKY AJ 1971 ECOLOGICAL MONOGRAPHS, MCMAHON RF 1975
27 3.333 4 5 EDWARDS NT 1977 ECOLOGY, CHAPMAN SB 1979 JOURNAL OF ECOLOGY, SCHWARTZKOPF SH
28 3.333 4 5 BACH C 1976 ECOLOGY, FOTHERINGHAM N 1976 ECOLOGY, VANCE RR 1972 ECOLOGY, CHIL
29 3.333 4 5 SUMMERFI.RJ 1972 JOURNAL OF ECOLOGY, SUMMERFI.RJ 1973 JOURNAL OF ECOLOGY, S
30 3.25 9 13 ANDERSON DC 1976 ECOLOGY, BROWN AV 1978 ECOLOGY, GRAHAME J 1973 JOURNAL OF AN
31 3 3 4 COCK MJW 1978 JOURNAL OF ANIMAL ECOLOGY, COOK RM 1978 JOURNAL OF ANIMAL ECOLOGY,
32 3 3 4 LEWIS WM 1974 ECOLOGICAL MONOGRAPHS, GANF GG 1974 JOURNAL OF ECOLOGY, TALLING JF
33 3 3 3 HELLER HC 1971 ECOLOGY, MEREDITH DH 1977 ECOLOGY, SHEPPARD DH 1971 ECOLOGY
34 3 3 3 WHITE DA 1978 ECOLOGY, HOPKINSON CS 1978 ECOLOGY, STIVEN AE 1979 ECOLOGICAL MONOG
35 3 3 3 REINERS WA 1979 ECOLOGY, NICHOLSON SA 1979 ECOLOGY, SICCAMA TG 1974 ECOLOGICAL MO
36 3 3 3 RICHERT SE 1973 JOURNAL OF ANIMAL ECOLOGY, ALLEN TF 1973 JOURNAL OF ECOLOGY, RIEC
37 3 3 3 BARBOUR MG 1974 ECOLOGY, OECHEL WC 1972 ECOLOGICAL MONOGRAPHS, ODENING WR 1974 EC
38 3 3 4 THOMAS JD 1974 JOURNAL OF ANIMAL ECOLOGY, THOMAS JD 1975 JOURNAL OF ANIMAL ECOLOG
39 3 3 3 TAFFE CA 1976 JOURNAL OF ANIMAL ECOLOGY, FREEMAN BE 1976 JOURNAL OF ANIMAL ECOLOG
40 3 3 4 GILL DE 1979 ECOLOGY, WILBUR HM 1977 ECOLOGY, COLLINS JP 1979 ECOLOGY
41 3 3 3 GOCHENAU.SE 1974 ECOLOGY, WICKLOW DT 1973 ECOLOGY, WICKLOW DT 1974 ECOLOGY
42 3 3 3 HARPER G 1976 JOURNAL OF ANIMAL ECOLOGY, WHITTAKE.JB 1971 JOURNAL OF ANIMAL ECOL
43 3 3 3 AYYAD MA 1974 JOURNAL OF ECOLOGY, AYYAD MA 1974 ECOLOGY, AYYAD MA 1973 JOURNAL OF
44 3 3 3 MOORE WS 1975 ECOLOGY, MCKAY FE 1971 ECOLOGY, MOORE WS 1971 ECOLOGY
45 3 3 4 REDFIELD AC 1972 ECOLOGICAL MONOGRAPHS, LINTHURST RA 1978 ECOLOGY, MAHALL BE 1976
46 3 3 3 KETTERSON ED 1976 ECOLOGY, FUENTES ER 1979 ECOLOGY, MCNAB BK 1971 ECOLOGY
47 3 3 3 ROCKWOOD LL 1974 ECOLOGY, ROCKWOOD LL 1973 ECOLOGY, ROCKWOOD LL 1976 ECOLOGY
48 3 3 3 HOOGLAND JL 1976 ECOLOGICAL MONOGRAPHS, VICTORIA JK 1973 ECOLOGY, COLLIAS NE 1971
49 3 3 3 COUGHLAN SJ 1977 JOURNAL OF ECOLOGY, HANSON RB 1979 ECOLOGY, CRAWFORD CC 1974 ECO
50 3 3 3 SPRINGETT JA 1977 JOURNAL OF ANIMAL ECOLOGY, LATTER PM 1978 JOURNAL OF ANIMAL ECO
51 3 3 5 JONES R 1973 JOURNAL OF ECOLOGY, JONES R 1972 JOURNAL OF ECOLOGY, JONES R 1975 JO
52 3 3 3 BAHR LM 1976 ECOLOGY, GALLAGHE.JL 1973 ECOLOGY, SMITH KL 1973 ECOLOGY
53 3 3 3 BAZZAZ FA 1974 ECOLOGY, RAYNAL DJ 1975 ECOLOGY, WIELAND NK 1975 ECOLOGY
54 2.8 6 9 BATES JW 1975 JOURNAL OF ECOLOGY, HUBALEK Z 1978 ECOLOGY, MADGWICK HA 1972 JOURNA
55 2.571 8 9 BISHOP JA 1976 JOURNAL OF ANIMAL ECOLOGY, BISHOP JA 1972 JOURNAL OF ANIMAL EC
56 2 2 3 ELLIOTT JM 1976 JOURNAL OF ANIMAL ECOLOGY, ELLIOTT JM 1975 JOURNAL OF ANIMAL ECOL
57 2 2 3 BERNSTEIN RA 1979 JOURNAL OF ANIMAL ECOLOGY, KING TJ 1977 JOURNAL OF ECOLOGY
Line 1 Col 1
(none)
Unicode (UTF-8)
Unix (LF)
Last saved: 2/28/14 1:18:34 PM
14,405 / 2,396 / 68

MCODE sets three node attributes:

- `MCODE_Cluster` contains the name of the cluster to which each node belongs.
- `MCODE_Score` indicates how strongly the neighbors around a node cluster together. This is similar to the Local clustering coefficient
- `MCODE_Node_Status` indicates whether a node is clustered, unclustered, or a seed node. Seed nodes are the reference nodes chosen by MCODE at the start of the cluster-detection process.

In the visualization below, node fill color is mapped to `MCODE_Cluster`. Node size is mapped to `MCODE_Node_Status`: unclustered nodes are small, seed nodes are large, and clustered nodes are intermediate in size.



Bibliographic Coupling over Time

If your dataset includes papers published over a long period of time, you may wish to analyze your bibliographic coupling graph as a dynamic network. This can give a visual impression of how fields and subfields evolve over time, in terms of whether they do or do not share cited references.

Command-line

Run the `graph` step again, but this time remove the `--merged` flag. This will create a separate graph from each of the data subsets created in the `slice` step.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --graph \
> --node-type=paper --graph-type=bibliographic_coupling --threshold=0.05 --weighted \
> --node-attr=date,jtitle,atitle
----------------------------------------
```

```
        Workflow step: Graph
----------------------------------------
Loading DataCollection with slices from /tmp/exampleID_DataCollection_sliced.pickle...done.
Using first slice in DataCollection: date.
Building author graph using coauthors method...done in 0.291323900223 seconds.
Saving GraphCollection to /tmp/exampleID_GraphCollection.pickle...done.
Writing graph summaries to /Users/erickpeirson/exampleOutput/exampleID_graphs.csv...done.
```

Re-run the `write` step. Use `--write-format xgmml` to select the dynamic XGMML export option.

```
$ tethne -I exampleID -O /Users/erickpeirson/exampleOutput --write --write-format xgmml
----------------------------------------
        Workflow step: Write
----------------------------------------
Loading GraphCollection from /tmp/exampleID_GraphCollection.pickle...done.
Writing graphs to /Users/erickpeirson/exampleOutput with format xgmml...done.
```

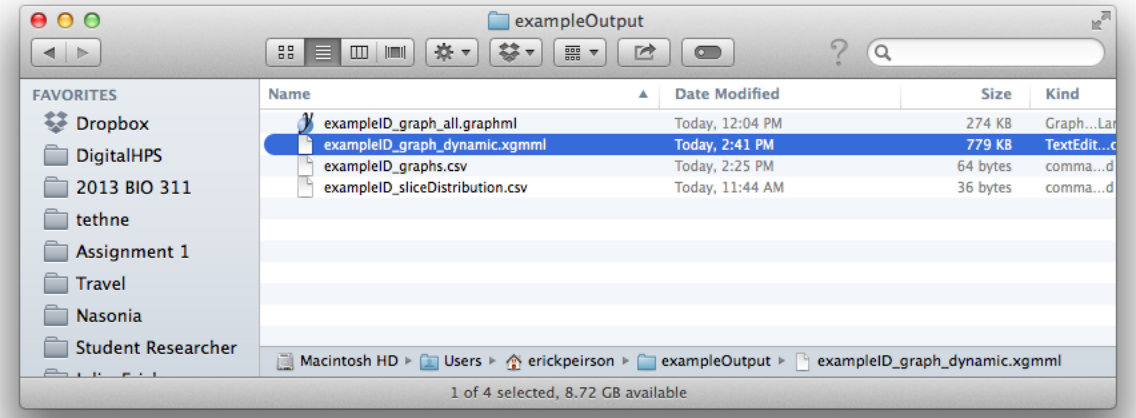
This should create a new file called `[DATASET_ID]_graph_dynamic.xgmml` in your output folder.

TethneGUI

Use the `< Back` button to return to `Step 3: Build Graphs`. Uncheck the `Ignore DataCollection slicing` option, and then click the `Build graph` button again. Then click `Next >`.

Skip the `analyze` step.

At the `write` step, select `xgmml` in the `Output format` menu, and click `Write graph(s)`. This should create a new file called `[DATASET_ID]_graph_dynamic.xgmml` in your output folder.



Python

Use the `paperCollectionBuilder` to build a `GraphCollection` from your `DataCollection`.

```
>>> from tethne.builders import paperCollectionBuilder
>>> builder = paperCollectionBuilder(D)
>>> C = builder.build('date', 'bibliographic_coupling', threshold=5,
...                   node_attribs=['date','jtitle','atitle'])
```

Use the `writers.collection.to_dxgmml()` method to create dynamic XGMML.

```
>>> import tethne.writers as wr
>>> wr.collection.to_dxgmml(C, "[OUTPUT_PATH]")
```

`[OUTPUT_PATH]` should be a path to the XGMML file that Tethne will create.

Visualization

See *Visualizing a dynamic network in Cytoscape* for instructions about how to visualize your dynamic network in Cytoscape (the parts about `attachment_probability` don't apply).

1.2.6 Step-By-Step Guide (Python)

Creating Networks from Bibliographic Data

Ready to Proceed?

Once you have collected your bibliographic data, you're ready to start building networks.

Parsing Data

Methods for parsing bibliographic data are contained in the `readers` module. Tethne parses bibliographic data into a list of `Paper` objects that can then be used to generate networks.

Many (but not all) of the networks that Tethne can generate require citation data. The current version of Tethne only supports citation data from the Web of Science, which can be parsed using the `readers.wos` module. For example:

```
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/to/savedrecs.txt")
```

Tethne can also parse data from JSTOR's Data-for-Research portal, using the `readers.dfr` module. Those data can be merged with a WoS dataset (see `readers.merge()`), or used on their own to generate coauthor networks, with `networks.authors.coauthors()`.

```
>>> import tethne.readers as rd
>>> papers = rd.dfr.read("/Path/to/DfR")
```

Creating Networks

There are many different network models that can be used to describe bibliographic data. These can be roughly divided into two categories: networks that describe relationships among documents, and networks that describe relationships among the authors of those documents. For specific methods, see *Networks of Documents* and *Networks of Authors*.

All network-building methods can be found in the `networks` module. `nt` is the recommended namespace convention.

```
>>> import tethne.networks as nt
```

There are two main ways of using network-building methods:

Generating a single network directly from a list of `Paper` objects All methods in `tethne.networks` take lists of `Paper` as arguments. For example:

```
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/to/savedrecs.txt")
>>> import tethne.networks as nt
>>> BC = nt.papers.bibliographic_coupling(papers, threshold=2)
```

Generating a `GraphCollection` from a `DataCollection` This is useful in cases where you want to evaluate the evolution of network structure over time, or compare networks generated using subsets of your data.

To generate a time-variant `GraphCollection`, slice your `DataCollection` using the `date` field. In the example below, data are sliced using a 4-year sliding time-window (for details about slicing, see `tethne.data.DataCollection.slice()`).

```
>>> # Parse data.
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/To/FirstDataSet.txt")

>>> # Create a DataCollection, and slice it.
>>> from tethne.data import DataCollection, GraphCollection
>>> D = DataCollection(papers)
>>> D.slice('date', 'time_window', window_size=4)

>>> # Build a GraphCollection using a network from tethne.networks.
>>> from tethne.builders import authorCollectionBuilder
>>> builder = authorCollectionBuilder(D)
>>> C = builder.build('date', 'coauthors')
```

`C.keys()` should now yield a list of publication dates in the original dataset.

A `DataCollection` can be sliced using any `int` or `str` field in the `Paper` class. If you wish to compare networks generated from two WoS downloads, for example, you could slice using the `accession` id:

```
>>> # Parse data.
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/To/FirstDataSet.txt")
>>> papers += rd.wos.read("/Path/To/SecondDataSet.txt")

>>> # Create a DataCollection, and slice it.
>>> from tethne.data import DataCollection, GraphCollection
>>> D = DataCollection(papers)
>>> D.slice('accession')

>>> # Build a GraphCollection using a network from tethne.networks.
>>> from tethne.builders import authorCollectionBuilder
>>> builder = paperCollectionBuilder(D)
>>> C = builder.build('date', 'cocitation', threshold=2)
```

`C.keys()` should now yield two values, each an accession UUID.

Networks of Documents

Methods for building networks in which vertices represent documents are provided in the `networks.papers` module.

tethne.networks.papers.author_coupling(papers)	Vertices are papers and edges indicates shared authorsh
tethne.networks.papers.bibliographic_coupling(papers)	Generate a bibliographic coupling network.
tethne.networks.papers.cocitation(papers[, ...])	Generate a cocitation network.
tethne.networks.papers.direct_citation(papers)	Create a traditional directed citation network.

Networks of Authors

Methods for building networks in which vertices represent authors are provided in the networks.authors module.

tethne.networks.authors.author_cocitation(papers)	Generates an author co-citation network; edges indicate c
tethne.networks.authors.author_coinstitution(Papers)	Generate a co-institution graph, where edges indicate sha
tethne.networks.authors.author_institution(Papers)	Generate a bi-partite graph connecting authors and their i
tethne.networks.authors.author_papers(papers)	Generate an author_papers network NetworkX directed g
tethne.networks.authors.coauthors(papers[, ...])	Generate a co-author network.

Analyzing Bibliographic Networks

All networks in Tethne are NetworkX Graphs. This means means that you can use the rich suite of algorithms provided by NetworkX to analyze your bibliographic networks.

Analyzing individual networks

If you built your network directly from a list of Paper, you can import and use NetworkX directly.

To calculate the betweenness-centrality of all of the nodes in a bibliographic coupling network, for example, use:

```
>>> # Parse your data:
>>> import tethne.readers as rd
>>> wos_list = rd.wos.parse_wos("/Path/to/savedrecs.txt")
>>> papers = rd.wos.rad(wos_list)

>>> # Build a bibliographic coupling network:
>>> import tethne.networks as nt
>>> BC = nt.papers.bibliographic_coupling(papers)

>>> # Use the NetworkX betweenness-centrality algorithm:
>>> import networkx as nx
>>> btw = nx.betweenness_centrality(G)
>>> btw
{'a': 0.0, 'c': 0.0, 'b': 0.6666666666666666, 'd': 0.0}
```

To add the betweenness-centrality values to your network as node attributes...

```
>>> nx.set_node_attributes(BC, 'betweenness', btw)
```

You can find a complete list of graph analysis algorithms in the NetworkX documentation.

A few additional methods internal to Tethne can be found in the analyze.graph module.

Analyzing a GraphCollection

The analyze.collection sub-package provides mechanisms for analyzing an entire GraphCollection. Most NetworkX algorithms are accessible via analyze.collection.algorithm(). To calculate betweenness centrality for an entire GraphCollection, for example, use:

```
>>> import tethne.analyze as az
>>> BC = az.collection.algorithm(C, 'betweenness_centrality')
>>> print BC[0]
{1999: 0.010101651117889644,
2000: 0.0008689093723107329,
2001: 0.010504898852426189,
2002: 0.009338654511194512,
2003: 0.007519105636349891}
```

For more information, see the analyze.collection sub-package.

tethne.analyze.collection.algorithm(C, ...)	Apply NetworkX method to each C
tethne.analyze.collection.connected(C, ...)	Performs analysis methods from ne
tethne.analyze.collection.edge_history(C, ...)	Returns a dictionary of attribute va
tethne.analyze.collection.node_history(C, ...)	Returns a dictionary of attribute va
tethne.analyze.collection.node_global_closeness_centrality(C, node)	Calculates global closeness central

Methods

1.3 Command-line Options

Invoke Tethne with python [TETHNE_PATH]. To find the [TETHNE PATH], start the Python interpreter in Terminal, and import Tethne. Then call tethne.__file__. In the example below, the path that we're looking for is /anaconda/lib/python2.7/site-packages/tethne:

```
$ python
Python 2.7.5 |Anaconda 1.6.1 (x86_64)| (default, Jun 28 2013, 22:20:13)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tethne
>>> tethne.__file__
'//anaconda/lib/python2.7/site-packages/tethne/__init__.pyc'
>>>
```

If you are using a Mac, we recommend creating an *Alias*. You can then call Tethne with:

```
$ tethne --help
```

1.3.1 Universal arguments

The following arguments should be included each time you run Tethne. Tethne uses --dataset-id to track your dataset through the workflow, so it should remain the same in each workflow step.

Argument	Alternative	Description
-I DATASET_ID	--dataset-id=DATASET_ID	Unique ID (required).
-t TEMP_DIR	--temp-dir=TEMP_DIR	Directory for storing temporary files (optional; default is /tmp).
-O OUTPATH	--outpath=OUTPATH	Path to save workflow output. Some workflow steps will generate summary statistics or other output.

A base pattern for calling Tethne might look like:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results
```

This will cause all output to be saved in the `results` folder inside your home directory.

1.3.2 Workflow steps

There are 5 steps in the workflow, each with a distinct set of arguments. These should be called sequentially (only the `--analyze` step can be skipped).

- *Read*
- *Slice*
- *Graph*
- *Analyze*
- *Write*

Read

Parses bibliographic data. There are two ways to do this:

Argument	Description
--read-file	Read from a single data file. Requires --data-path and --data-format.
--read-dir	Read from a directory containing multiple data files. Requires --data-path and --data-format.

The following arguments are also required:

Argument	Alternative	Description
-P DATAPATH	--data-path=DATAPATH	Full path to dataset.
-F DATAFORMAT	--data-format=DATAFORMAT	Format of input dataset (WOS, DFR).

For example:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results --read-file \
> -P /path/to/your\ data/download.txt -F WOS
```

Resulting in something like:

```
----------------------------------------
        Workflow step: Read
----------------------------------------
Reading WOS data from file /path/to/your data/download.txt...done.
Read 500 papers in 1.69956803 seconds. Accession: 19825ab7-6176-4742-8cf2-0093d751b5f3.
Generating a new DataCollection...done.
Saving DataCollection to /tmp/fundata01_DataCollection.pickle...done.
```

WOS: Web of Science field-tagged format; **DFR:** JSTOR Data-for-Research dataset in XML format.

Slice

Slicing divides your dataset up along one or more axes (a key in the `Paper` class) for analysis. This prepared your dataset for comparative analysis in later steps. You might wish to, for example, analyze your dataset diachronically by slicing by `date`, or you might wish to compare data from different journals by slicing by `jtitle`. Use `accession` if you wish to compare data from different data files.

As of `v.0.3`, slicing is limited to `date`, `jtitle`, and `accession`.

Argument	Description
`--slice`	**Slice your dataset for comparison along a key axis.** Requires `--slice-axis`. If `--outpath` is set, produces a table with binned paper frequencies in `[OUTPATH]/[DATASET_ID]_slices.csv`.

The following arguments are required:

Argument	Alternative	Description
`-S SLICE_AXIS`	`--slice-axis=SLICE_AXIS`	Key along which to slice the dataset. This can be any of the fields listed in `Paper`.
`-M SLICE_METHOD`	`--slice-method=SLICE_METHOD`	Method used to slice `DataCollection`. Available methods: `time_window`, `time_period`. For details, see `DataCollection.slice()`. Default is time_period.

The following arguments are optional:

Argument	Description
`--slice-window-size=SIZE`	Size of slice time-window or period, in years. Default: 1.
`--slice-step-size=SIZE`	Amount to advance time-window in each step (ignored for time-period).
`--cumulative`	If True, the data from each successive slice includes the data from all preceding slices.

For example:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results --slice \
>  -S date,jtitle -M time_window --slice-window-size=4 --slice-step-size=1 --cumulative
```

Resulting in something like:

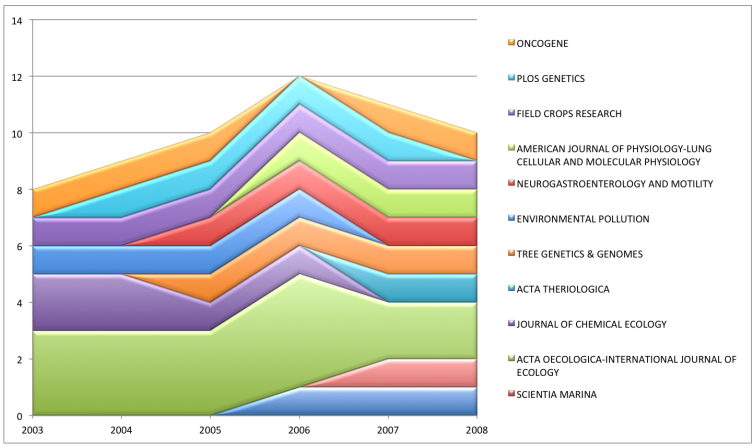
```
----------------------------------------
    Workflow step: Slice
----------------------------------------
Loading DataCollection from /tmp/fundata01_DataCollection.pickle...done.
Slicing DataCollection by date...done.
Slicing DataCollection by jtitle...done.
Saving slice distribution to ~/results/fundata01_sliceDistribution.csv...done.
Saving sliced DataCollection to /tmp/fundata01_DataCollection_sliced.pickle...done.
```

`~/results/fundata01_sliceDistribution.csv` contains a comma-separated table with the distribution of papers across `date` and `jtitle`. For example:

Table 1.4: fundata01_sliceDistribution.csv

	2003	2004	2005	2006	2007	2008
ENVIRONMENTAL BIOLOGY OF FISHES	0.0	0.0	0.0	1.0	1.0	1.0
SCIENTIA MARINA	0.0	0.0	0.0	0.0	1.0	1.0
ACTA OECOLOGICA-INTERNATIONAL JOURNAL OF ECOLOGY	3.0	3.0	3.0	4.0	2.0	2.0
JOURNAL OF CHEMICAL ECOLOGY	2.0	2.0	1.0	1.0	0.0	0.0
ACTA THERIOLOGICA	0.0	0.0	0.0	0.0	1.0	1.0
TREE GENETICS & GENOMES	0.0	0.0	1.0	1.0	1.0	1.0
ENVIRONMENTAL POLLUTION	1.0	1.0	1.0	1.0	0.0	0.0
NEUROGASTROENTEROLOGY AND MOTILITY	0.0	0.0	1.0	1.0	1.0	1.0
FIELD CROPS RESEARCH	1.0	1.0	1.0	1.0	1.0	1.0
PLOS GENETICS	0.0	1.0	1.0	1.0	1.0	0.0
ONCOGENE	1.0	1.0	1.0	0.0	1.0	1.0

You can easily visualize these data using your favorite spreadsheet software.



Graph

Argument	Description
`--graph`	Generate a graph (or collection of graphs). If `--outpath` is set, produces a table with the number of nodes and edges per graph in `[OUTPATH]/[DATASET_ID]_graphs.csv`.

The following arguments should be used:

Argument	Alternative	Description
`-N NODE_TYPE`	`--node-type=NODE_TYPE`	Must be one of: `author`, `paper`.
`-T GRAPH_TYPE`	`--graph-type=GRAPH_TYPE`	Name of a network-builing method. Can be one of any of the methods listed in `networks`. e.g. if `-n` is `author`, `-t` could be `coauthors`.

Available network-building methods (as of v0.3.0-alpha)

Node Type	Graph Type	Method
paper	author_coupling	tethne.networks.papers.author_coupling()
paper	bibliographic_coupling	tethne.networks.papers.bibliographic_coupling()
paper	cocitation	tethne.networks.papers.cocitation()
paper	direct_citation	tethne.networks.papers.direct_citation()
author	author_cocitation	tethne.networks.authors.author_cocitation()
author	author_coinstitution	tethne.networks.authors.author_coinstitution()
author	author_papers	tethne.networks.authors.author_papers()
author	coauthors	tethne.networks.authors.coauthors()

If you have sliced your data in a previous step, but wish to generate a network based on the entire dataset, you may use:

Argument	Description
--merged	Ignore DataCollection slicing, and build a single graph from all Papers.

Some methods use additional keyword arguments that affect the resulting graph. The following arguments can be used to set common keyword arguments. The meaning of these arguments varies between methods; consult networks for descriptions of each network-building method.

Argument	Description
--threshold=THRESHOLD	Set the 'threshold' argument. Applies to all except: direct_citation, author_institution, author_papers.
--topn=TOPN	Set the 'topn' argument. Applies to: cocitation.
--node-attr=NODE_ATTR	List of attributes to include for each node. e.g. --node-attr=date,atitle,jtitle. Applies to: all.
--edge-attr=EDGE_ATTR	List of attributes to include for each edge. e.g. --edge-attr=ayjid,atitle,date. Applies to: all.
--node-id=NODE_ID	Field to use as node id (for papers graphs). e.g. --node-id=ayjid. Applies to: all paper methods.
--weighted	Trigger the 'weighted' argument. Applies to: bibliographic_coupling.

For example:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results --graph \
> -N author -T coauthors --edge-attr=ayjid,date,jtitle
```

Resulting in something like:

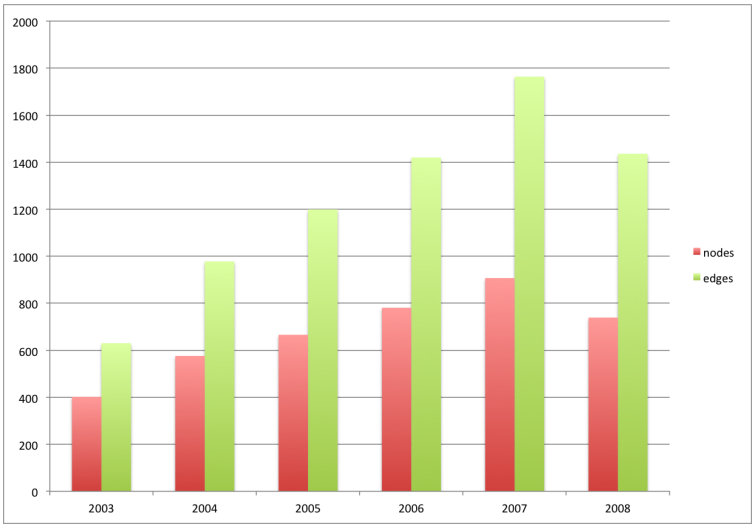
```
----------------------------------------
        Workflow step: Graph
----------------------------------------
Loading DataCollection with slices from /tmp/fundata01_DataCollection_sliced.pickle...done.
Using first slice in DataCollection: date.
Building author graph using coauthors method...done in 0.426736116409 seconds.
Saving GraphCollection to /tmp/fundata01_GraphCollection.pickle...done.
Writing graph summaries to ~/results/fundata01_graphs.csv...done.
```

~/results/fundata01_graphs.csv contains a comma-separated table with the number of nodes and edges per graph (indexed by date, in this case). For example:

Table 1.5: fundata01_graphs.csv

index	nodes	edges
2003	402	630
2004	576	978
2005	666	1199
2006	781	1420
2007	907	1764
2008	739	1436

You can easily visualize these data using your favorite spreadsheet software.



Analyze

The analysis workflow step is optional. As of v0.3.0-alpha, `--analyze` triggers `analyze.collection.algorithm()`, which calls a graph analysis algorithm in NetworkX. So far this has been tested for centrality algorithms only.

Argument	Description
`--analyze`	Analyze a graph (or collection of graphs). If `--outpath` is set, produces a table with the mean and variance of the algorithm result for each graph, in `[OUTPATH]/[DATASET_ID]_[ALGORITHM]_analysis.csv`.

Use the `--algorithm` argument to select an algorithm. This should be the name of an method in the 'NetworkX centrality algorithm methods <http://networkx.github.io/documentation/latest/reference/algorithms.centrality.html>'+_.

Argument	Alternative	Description
`-A ALGORITHM`	`--algorithm=ALGORITHM`	Name of a NetworkX graph analysis algorithm.

For example:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results --analyze \
> -A betweenness_centrality
```

Resulting in something like:

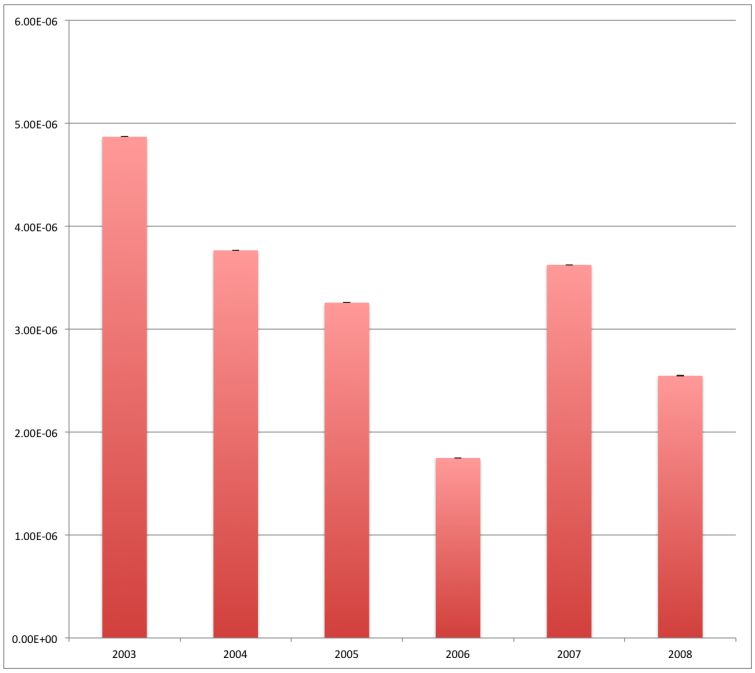
```
----------------------------------------
        Workflow step: Analyze
----------------------------------------
Loading GraphCollection from /tmp/fundata01_GraphCollection.pickle...done.
Analyzing GraphCollection with betweenness_centrality...done.
Writing graph analysis results to ~/results/fundata01_betweenness_centrality_analysis.csv...done.
Saving GraphCollection to /tmp/fundata01_GraphCollection.pickle...done.
```

`~/results/fundata01_betweenness_centrality_analysis.csv` contains a comma-separated table with the mean and variance of per-node betweenness-centrality for each graph (indexed by `date`, in this case). For example:

Table 1.6: fundata01_betweenness_centrality_analysis.csv

index	mean	variance
2003	4.8696666294462848e-06	5.8521060847378354e-10
2004	3.7662643707182407e-06	5.0172975473198493e-10
2005	3.2576284954217712e-06	4.8988211004664643e-10
2006	1.749020895995168e-06	1.5297987419477807e-10
2007	3.6252321590741106e-06	1.4535627363876269e-09
2008	2.5476025127262327e-06	1.0268407559417709e-09

You can easily visualize these data using your favorite spreadsheet software.



Write

You can visualize networks using software like Cytoscape or Gephi. The writing workflow step involves converting a collection of NetworkX graphs into a structured graph file. Tethne can generate both static and dynamic networks. If a static network format is chosen, each graph in the collection will be written to a separate file.

Argument	Description
`--write`	Write a graph (or collection of graphs) to a structured format, in `[OUTPATH]`.

The `--write-format` argument is required:

Argument	Alternative	Description
-W WRITE_FORMAT	–write-format=FORMAT	Output format for graph(s). If a static graph format is chosen (e.g. graphml), each slice in the GraphCollection will result in a separate file. Supported writers: (static) graphml; (dynamic) xgmml.

For example:

```
$ python ~/Downloads/tethne-python/tethne -I fundata01 -O ~/results --write \
> -W graphml
```

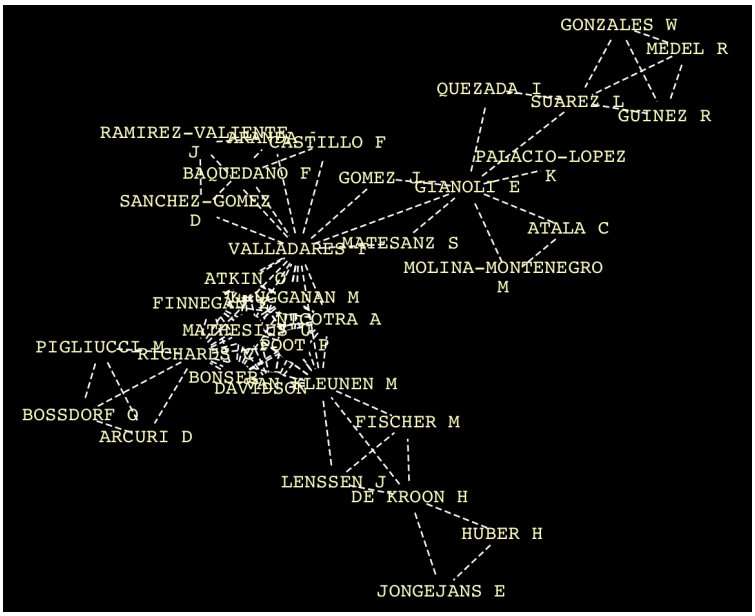
Resulting in something like:

```
----------------------------------------
        Workflow step: Write
----------------------------------------
Loading GraphCollection from /tmp/fundata01_GraphCollection.pickle...done.
Writing graphs to ~/results with format graphml...done.
```

And generating the following files in `~/results`:

```
-rw-r--r--  1 erickpeirson  staff  196686 Feb 23 10:52 fundata01_graph_2003.graphml
-rw-r--r--  1 erickpeirson  staff  299885 Feb 23 10:52 fundata01_graph_2004.graphml
-rw-r--r--  1 erickpeirson  staff  359992 Feb 23 10:52 fundata01_graph_2005.graphml
-rw-r--r--  1 erickpeirson  staff  427821 Feb 23 10:52 fundata01_graph_2006.graphml
-rw-r--r--  1 erickpeirson  staff  515779 Feb 23 10:52 fundata01_graph_2007.graphml
-rw-r--r--  1 erickpeirson  staff  418702 Feb 23 10:52 fundata01_graph_2008.graphml
```

Opening `fundata01_graph_2007.graphml` in Cytoscape yields something like:



1.4 tethne Package

1.4.1 `tethne` Package

Tethne is a package for analyzing citation data from the Web of Science. Modules within Tethne can generate a variety of networks, such as bibliographic coupling, citation, author-paper, and co-author networks, using networkx.

tethne.analyze	The tethne.analyze sub-package provides additional analysis methods not
tethne.builders	Classes for building a GraphCollection .
tethne.data	Classes for handling bibliographic data.
tethne.matrices	Methods for generating matrices from Paper objects and other data.
tethne.networks	Methods for building networks from bibliographic data.
tethne.readers	Methods for parsing bibliographic datasets.
tethne.utilities	Helper functions for tethne.networks .
tethne.writers	Export networks to structured and unstructured formats, for visualization.

1.4.2 __main__ Module

tethne.__main__.**isfloat**(*x*)

tethne.__main__.**isint**(*x*)

1.4.3 builders Module

Classes for building a GraphCollection .

builder(D)	Base class for builders.
authorCollectionBuilder(D)	Builds a GraphCollection with method in
paperCollectionBuilder(D)	Builds a GraphCollection with method in

class tethne.builders.**authorCollectionBuilder**(*D*)

Bases: tethne.builders.builder

Builds a GraphCollection with method in tethne.networks.authors from a DataCollection .

Methods

build(graph_axis, graph_type, **kwargs)	Generates graphs for each slice along graph_axis in

build(*graph_axis*, *graph_type*, ***kwargs*)

Generates graphs for each slice along graph_axis in DataCollection D.

Other axes in D are treated as attributes.

Usage

```
>>> import tethne.readers as rd
>>> data = rd.wos.read("/Path/to/wos/data.txt")
>>> from tethne.data import DataCollection
>>> D = DataCollection(data) # Indexed by wosid, by default.
>>> D.slice('date', 'time_window', window_size=4)
>>> from tethne.builders import authorCollectionBuilder
>>> builder = authorCollectionBuilder(D)
>>> C = builder.build('date', 'coauthors')
>>> C
<tethne.data.GraphCollection at 0x104ed3550>
```

class tethne.builders.**builder**(*D*)
Bases: object

Base class for builders.

class tethne.builders.**paperCollectionBuilder**(*D*)
Bases: tethne.builders.builder

Builds a GraphCollection with method in tethne.networks.papers from a DataCollection .

Methods

build(graph_axis, graph_type, **kwargs)	Generates graphs for each slice along graph_axis in

build(*graph_axis*, *graph_type*, ***kwargs*)
Generates graphs for each slice along graph_axis in DataCollection D.

Other axes in D are treated as attributes.

Usage

```
>>> import tethne.readers as rd
>>> data = rd.wos.read("/Path/to/wos/data.txt")
>>> from tethne.data import DataCollection
>>> D = DataCollection(data) # Indexed by wosid, by default.
>>> D.slice('date', 'time_window', window_size=4)
>>> from tethne.builders import paperCollectionBuilder
>>> builder = paperCollectionBuilder(D)
>>> C = builder.build('date', 'bibliographic_coupling', threshold=2)
>>> C
<tethne.data.GraphCollection at 0x104ed3550>
```

1.4.4 data Module

Classes for handling bibliographic data.

Paper()	Base class for Papers.
DataCollection(data[, index_by])	A DataCollection organizes Papers for analysis.
GraphCollection()	Collection of NetworkX nx.classes.graph.Graph objects,
LDAModel(doc_topic, top_word, top_keys, ...)	Organizes parsed output from MALLET's LDA modeling algorithm.

class tethne.data.**DataCollection**(*data*, *index_by='wosid'*)
Bases: object

A DataCollection organizes Papers for analysis.

The DataCollection is initialized with some data, which is indexed by a key in Paper (default is wosid). The DataCollection can then be sliced (DataCollection.slice()) by other keys in Paper .

Usage

```
>>> import tethne.readers as rd
>>> data = rd.wos.read("/Path/to/wos/data.txt")
>>> data += rd.wos.read("/Path/to/wos/data2.txt")    # Two accessions.
>>> from tethne.data import DataCollection
```

```
>>> D = DataCollection(data) # Indexed by wosid, by default.
>>> D.slice('date', 'time_window', window_size=4)
>>> D.slice('accession')
>>> D
<tethne.data.DataCollection at 0x10af0ef50>
```

Methods

N_axes()	Returns the number of slice axes for this DataCollection .
distribution()	Returns a Numpy array describing the number of Paper
distribution_2d(x_axis, y_axis)	Returns a Numpy array describing the number of Paper
get_axes()	Returns a list of all slice axes for this DataCollection .
get_by(key_indices[, papers])	Given a set of (key, index) tuples, return the corresponding subset of
get_slice(key, index[, papers])	Yields a specific slice.
get_slices(key[, papers])	Yields slices for key.
indices()	Yields a list of indices of all papers in this DataCollection
papers()	Yield the complete set of Paper instances in this
slice(key[, method])	Slices data by key, using method (if applicable).

N_axes()
Returns the number of slice axes for this DataCollection .

distribution()
Returns a Numpy array describing the number of Paper associated with each slice-coordinate.

WARNING: expensive for a DataCollection with many axes or long axes. Consider using distribution_2d() .

Returns **dist** : Numpy array

An N-dimensional array. Axes are given by DataCollection.get_axes() and values are the number of Paper at that slice-coordinate.

Raises **RuntimeError** : DataCollection has not been sliced.

distribution_2d(*x_axis*, *y_axis*)
Returns a Numpy array describing the number of Paper associated with each slice-coordinate, for x and y axes spcified.

Returns **dist** : Numpy array

A 2-dimensional array. Values are the number of Paper at that slice-coordinate.

Raises **RuntimeError** : DataCollection has not been sliced.

KeyError: Invalid slice axes for this DataCollection. :

get_axes()
Returns a list of all slice axes for this DataCollection .

get_by(*key_indices*, *papers=False*)
Given a set of (key, index) tuples, return the corresponding subset of Paper indices (or Paper instances themselves, if papers is True).

Parameters **key_indices** : list

A list of (key, index) tuples.

Returns **plist** : list

A list of paper indices, or Paper instances.

Raises **RuntimeError** : DataCollection has not been sliced.

get_slice(*key*, *index*, *papers=False*)
Yields a specific slice.

Parameters **key** : str

Key from Paper that has previously been used to slice data in this DataCollection .

index : str or int

Slice index for key (e.g. 1999 for 'date').

Returns **slice** : list

List of paper indices in this DataCollection , or (if papers is True) a list of Paper instances.

Raises **RuntimeError** : DataCollection has not been sliced.

KeyError : Data has not been sliced by [key]

KeyError : [index] not a valid index for [key]

get_slices(*key*, *papers=False*)
Yields slices for key.

Parameters **key** : str

Key from Paper that has previously been used to slice data in this DataCollection .

Returns **slices** : dict

Keys are slice indices. If papers is True, values are lists of Paper instances; otherwise returns paper indices (e.g. 'wosid').

Raises **RuntimeError** : DataCollection has not been sliced.

KeyError : Data has not been sliced by [key]

indices()
Yields a list of indices of all papers in this DataCollection

Returns **list** :

List of indices.

papers()
Yield the complete set of Paper instances in this DataCollection .

Returns **papers** : list

A list of Paper

slice(*key*, *method=None*, ***kwargs*)
Slices data by key, using method (if applicable).

Methods available for slicing a DataCollection:

Method	Description	Key	kwargs
time_window	Slices data using a sliding time-window. Dataslices are indexed by the start of the time-window.	date	window_size step_size
time_period	Slices data into time periods of equal length. Dataslices are indexed by the start of the time period.	date	window_size

The main difference between the sliding time-window (`time_window`) and the time-period (`time_period`) slicing methods are whether the resulting periods can overlap. Whereas time-period slicing divides data into subsets by sequential non-overlapping time periods, subsets generated by time-window slicing can overlap.

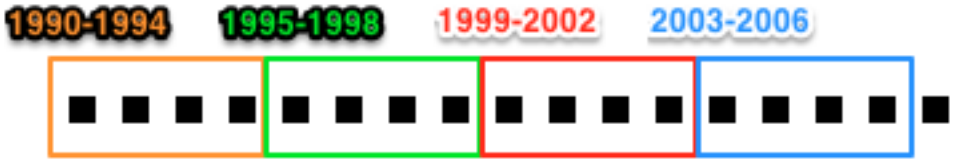


Figure 1.3: **Time-period** slicing, with a window-size of 4 years.

Figure 1.4: **Time-window** slicing, with a window-size of 4 years and a step-size of 1 year.

Avilable kwargs:

Argument	Type	Description
window_size	int	Size of time-window or period, in years (default = 1).
step_size	int	Amount to advance time-window or period in each step (ignored for time_period).
cumulative	bool	If True, the data from each successive slice includes the data from all preceding slices. Only applies if key is 'date' (default = False).

Parameters **key** : str

key in `Paper` by which to slice data.

method : str (optional)

Dictates how data should be sliced. See table for available methods. If key is 'date', default method is time_period with window_size and step_size of 1.

kwargs : kwargs

See methods table, above.

class `tethne.data.`**`GraphCollection`**

Bases: `object`

Collection of NetworkX `nx.classes.graph.Graph` objects, organized by some index (e.g. time).

A `GraphCollection` can be generated using classes in the `tethne.builders` module. See *Generating a GraphCollection from a DataCollection* for details.

Methods

`compose`()	Returns the simple union of all `Graph` in the
`edges`([overwrite])	Return complete set of edges for this `GraphCollection` .
`load`(filepath)	Loads a pickled (serialized) `GraphCollection` from filepath.
`nodes`([overwrite])	Return complete set of nodes for this `GraphCollection` .
`save`(filepath)	Pickles (serializes) the `GraphCollection` .

compose()

Returns the simple union of all `Graph` in the `GraphCollection` .

Returns **composed** : `Graph`

Simple union of all `Graph` in the `GraphCollection` .

Notes

Node or edge attributes that vary over slices should be ignored.

edges(*overwrite=False*)

Return complete set of edges for this `GraphCollection` .

If this method has been called previously for this `GraphCollection` then will not recompute unless overwrite = True.

Parameters **overwrite** : bool

If True, will generate new node list, even if one already exists.

Returns **edges** : list

List (complete set) of edges for this `GraphCollection` .

load(*filepath*)

Loads a pickled (serialized) `GraphCollection` from filepath.

Parameters **filepath** : string

Full path to pickled `GraphCollection` .

Raises **UnpicklingError** : Raised when there is some issue in unpickling.

IOError : File does not exist, or cannot be read.

nodes(*overwrite=False*)

Return complete set of nodes for this `GraphCollection` .

If this method has been called previously for this `GraphCollection` then will not recompute unless overwrite = True.

Parameters **overwrite** : bool

If True, will generate new node list, even if one already exists.

Returns **nodes** : list

List (complete set) of node identifiers for this GraphCollection .

save (*filepath*)
Pickles (serializes) the GraphCollection .

Parameters **filepath :** :

Full path of output file.

Raises **PicklingError** : Raised when unpicklable objects are Pickled.

IOError : File does not exist, or cannot be opened.

class tethne.data.**LDAModel** (*doc_topic*, *top_word*, *top_keys*, *metadata*, *vocabulary*)
Bases: object

Organizes parsed output from MALLET's LDA modeling algorithm.

Used by readers.mallet.

Methods

docs_in_topic(z[, topD])	Returns a list of the topD documents most representative of topic z.
topics_in_doc(d[, topZ])	Returns a list of the topZ most prominent topics in a document.

docs_in_topic (*z*, *topD=None*)
Returns a list of the topD documents most representative of topic z.

Parameters **z** : int

A topic index.

topD : int or float

Number of prominent topics to return (int), or threshold (float).

Returns **documents** : list

List of (document, proportion) tuples.

topics_in_doc (*d*, *topZ=None*)
Returns a list of the topZ most prominent topics in a document.

Parameters **d** : str or int

An identifier from a Paper key.

topZ : int or float

Number of prominent topics to return (int), or threshold (float).

Returns **topics** : list

List of (topic, proportion) tuples.

class tethne.data.**Paper**
Bases: object

Base class for Papers.

Behaves just like a dict, but enforces a limited vocabulary of keys, and specific data types.

The following fields (and corresponding data types) are allowed:

Field	Type	Description
aulast	list	Authors' last name, as a list.
auinit	list	Authors' first initial as a list.
institution	dict	Institutions with which the authors are affiliated.
atitle	str	Article title.
jtitle	str	Journal title or abbreviated title.
volume	str	Journal volume number.
issue	str	Journal issue number.
spage	str	Starting page of article in journal.
epage	str	Ending page of article in journal.
date	int	Article date of publication.
country	dict	Author-Country mapping.
citations	list	A list of `Paper` instances.
ayjid	str	First author's name (last fi), pubdate, and journal.
doi	str	Digital Object Identifier.
pmid	str	PubMed ID.
wosid	str	Web of Science UT fieldtag value.
accession	str	Identifier for data conversion accession.

None values are also allowed for all fields.

Methods

`authors`()	Returns a list of author names (FI LAST).
`iteritems`()	Returns an iterator for the `Paper`'s metadata fields
`keys`()	Returns the keys of the `Paper`'s metadata fields.
`values`()	Returns the values of the `Paper`'s metadata fields.

authors()
Returns a list of author names (FI LAST).

iteritems()
Returns an iterator for the `Paper`'s metadata fields

keys()
Returns the keys of the `Paper`'s metadata fields.

values()
Returns the values of the `Paper`'s metadata fields.

1.4.5 `workflow` Module

Methods for network analysis.

tethne.workflow.**closeness_introgression**(*papers*, *node*, *window_size*, *normalize=False*)
Analyzes the global closeness centrality of a node over time.

Parameters **papers** : list

A list of `Paper` instances.

node : any

Handle of the node to analyze.

window_size : int

Size of time-window.

normalize : bool

If True, normalizes global closeness centrality for each year against the average closeness centrality for that year. This will require substantially more processing time, and values will usually be >> 0.

Returns **trajectory** : dict

Global closeness centrality for node over specified period.

1.4.6 Subpackages

analyze Package

analyze Package

The tethne.analyze sub-package provides additional analysis methods not provided by NetworkX, as well as methods for using NetworkX algorithms on an entire GraphCollection.

collection	Methods for analyzing GraphCollection objects.
graph	Methods for network analysis.

collection Module

Methods for analyzing GraphCollection objects.

For the most part, these methods simply provide systematic access to algorithms in NetworkX.

tethne.analyze.collection.**algorithm**(*C*, *method*, ***kwargs*)

Apply NetworkX method to each Graph in GraphCollection.

Passes kwargs to specified NetworkX method for each Graph, and returns a dictionary of results indexed by element (node or edge) and graph index (e.g. date).

Parameters **C** : GraphCollection

The GraphCollection to analyze. The specified method will be applied to each Graph in **C**.

method : string

Name of a method in NetworkX to execute on graph collection.

****kwargs** :

A list of keyword arguments that should correspond to the parameters of the specified method.

Returns **results** : dict

A nested dictionary of results: results/elem(node or edge)/graph index.

Raises **ValueError** :

If name is not in networkx, or if no such method exists.

Examples

Betweenness centrality:

```
>>> import tethne.analyze as az
>>> BC = az.collection.algorithm(C, 'betweenness_centrality')
>>> print BC[0]
{1999: 0.010101651117889644,
2000: 0.0008689093723107329,
2001: 0.010504898852426189,
2002: 0.009338654511194512,
2003: 0.007519105636349891}
```

tethne.analyze.collection.**attachment_probability**(*C*)

Calculates the observed attachment probability for each node at each time-step.

Attachment probability is calculated based on the observed new edges in the next time-step. So if a node acquires new edges at time t, this will accrue to the node's attachment probability at time t-1. Thus at a given time, one can ask whether degree and attachment probability are related.

Parameters **C** : GraphCollection

Must be sliced by 'date'. See `GraphCollection.slice()`.

Returns **probs** : dict

Keyed by index in C.graphs, and then by node.

tethne.analyze.collection.**connected**(*C*, *method*, ***kwargs*)

Performs analysis methods from networkx.connected on each graph in the collection.

Parameters **C** : GraphCollection

The GraphCollection to analyze. The specified method will be applied to each Graph in **C**.

method : string

Name of method in networkx.connected.

****kwargs** : kwargs

Keyword arguments, passed directly to method.

Returns **results** : dictionary

Keys are graph indices, values are output of method for that graph.

Raises **ValueError** :

If name is not in networkx.connected, or if no such method exists.

Examples

```
>>> import tethne.data as ds
>>> import tethne.analyze as az
>>> import networkx as nx
>>> C = ds.GraphCollection()
>>> # Generate some random graphs
>>> for graph_index in xrange(1999, 2004):
>>>     g = nx.random_regular_graph(4, 100)
>>>     C[graph_index] = g
```

```
>>> results = az.collection.connected(C, 'connected', k=None)
>>> print results
{1999: False,
2000: False,
2001: False,
2002: False,
2003: False }
```

tethne.analyze.collection.**edge_history**(*C*, *source*, *target*, *attribute*, *verbose=False*)

Returns a dictionary of attribute vales for each Graph in C for a single edge.

Parameters **C** : GraphCollection

source : str

Identifier for source node.

target : str

Identifier for target node.

attribute : str

The attribute of interest; e.g. 'betweenness_centrality'

verbose : bool

If True, prints status and debug messages.

Returns **history** : dict

Keys are Graph keys in C; values are attribute values for edge.

tethne.analyze.collection.**node_global_closeness_centrality**(*C*, *node*)

Calculates global closeness centrality for node in each graph in GraphCollection C.

tethne.analyze.collection.**node_history**(*C*, *node*, *attribute*, *verbose=False*)

Returns a dictionary of attribute values for each Graph in C for a single node.

Parameters **C** : GraphCollection

node : str

The node of interest.

attribute : str

The attribute of interest; e.g. 'betweenness_centrality'

verbose : bool

If True, prints status and debug messages.

Returns **history** : dict

Keys are Graph keys in C; values are attribute values for node.

graph Module

Methods for network analysis.

tethne.analyze.graph.**global_closeness_centrality**(*g*, *normalize=True*)

Calculates global closeness centrality for all nodes in the network.

See node_global_closeness_centrality() for more information.

Parameters **g** : networkx.Graph

normalize : boolean

If True, normalizes centrality based on the average shortest path length. Default is True.

Returns **C** : dict

Dictionary of results, with node identifiers as keys and gcc as values.

tethne.analyze.graph.**node_global_closeness_centrality**(*g*, *node*, *normalize=True*)

Calculates the global closeness centrality of a single node in the network.

Closeness centrality is based on the average shortest path length between a focal node and all other nodes in the network. For multi-component graphs, conventional closeness centrality metrics fail because it is not possible to traverse between a given node and all other nodes in the graph. Global closeness centrality is calculated in a way that yields values even for multi-component graphs. For an example of how global closeness centrality can be used to analyze co-authorship networks, see the blog post here.

To calculate the global closeness centrality of a single node, try:

```
>>> import tethne.analyze as az
>>> ngbc = az.node_global_closeness_centrality(BC, 'LEE 1975 EVOLUTION')
>>> ngbc
0.154245
```

You can calculate the global closeness centrality of all nodes in the network using `global_closeness_centrality()`.

```
>>> GBC = az.global_closeness_centrality(BC)
>>> GBC
{'a': 0.0, 'c': 0.0, 'b': 0.6666666666666666, 'd': 0.0}
```

For connected graphs, this is equivalent to conventional betweenness centrality. For disconnected graphs, works around infinite path lengths between nodes in different components.

Parameters **g** : networkx.Graph

node : any

Identifier of node of interest in g.

normalize : boolean

If True, normalizes centrality based on the average shortest path length. Default is True.

Returns **c** : float

Global closeness centrality of node.

matrices Package

matrices Package

Methods for generating matrices from `Paper` objects and other data.

dfr	Methods for generating Numpy data objects from JSTOR Data-for-Research datasets.

dfr Module

Methods for generating Numpy data objects from JSTOR Data-for-Research datasets.

array(data[, normalize, verbose])	Yields a Numpy array, along with feature-index and document-index mappings.
matrix(data[, normalize, verbose])	Yields a Numpy matrix, along with feature-index and document-index mappings.

class tethne.matrices.dfr.**Map**
Bases: object

Maps integer indices to string values.

tethne.matrices.dfr.**array**(*data*, *normalize=False*, *verbose=False*)
Yields a Numpy array, along with feature-index and document-index mappings.

Usage

```
>>> import tethne.readers as rd
>>> import tethne.matrices as mt
>>> data = rd.dfr.ngrams("/Path/to/DfR/data")
>>> A, doc_index, feat_index = mt.dfr.array(data, normalize=True)
```

Parameters **data** : dict

Keys are document identifiers (e.g. DOIs), values are lists of feature- frequency tuples.

normalize : bool

If True, matrix values are relative to the maximum value in the matrix.

Returns **A** : Numpy array

Columns are documents, rows are features.

document_index : class:.*Map*

Maps column indices to document identifiers (keys of provided data).

feature_index : Map

Maps row indices to features.

tethne.matrices.dfr.**matrix**(*data*, *normalize=False*, *verbose=False*)
Yields a Numpy matrix, along with feature-index and document-index mappings.

Usage

```
>>> import tethne.readers as rd
>>> import tethne.matrices as mt
>>> data = rd.dfr.ngrams("/Path/to/DfR/data")
>>> M, doc_index, feat_index = mt.dfr.matrix(data, normalize=True)
```

Parameters **data** : dict

Keys are document identifiers (e.g. DOIs), values are lists of feature- frequency tuples.

normalize : bool

If True, matrix values are relative to the maximum value in the matrix.

Returns **M** : Numpy matrix

Columns are documents, rows are features.

document_index : class:.*Map*

Maps column indices to document identifiers (keys of provided data).

feature_index : `Map`

Maps row indices to features.

networks Package

networks Package

Methods for building networks from bibliographic data.

Each network relies on certain meta data in the `Paper` associated with each document. Often we wish to construct a network with nodes representing these documents and edges representing relationships between those documents, but this is not always the case.

Where it is the case, it is recommended but not required that nodes are represented by an identifier from {ayjid, wosid, pmid, doi}. Each has certain benefits. If the documents to be networked come from a single database source such as the Web of Science, wosid is most appropriate. If not, using doi will result in a more accurate, but also more sparse network; while ayjid will result in a less accurate, but more complete network.

Any type of meta data from the `Paper` may be used as an identifier, however.

We use "head" and "tail" nomenclature to refer to the members of a directed edge (x,y), x -> y, xy, etc. by calling x the "tail" and y the "head".

`authors`	Methods for generating networks in which authors are vertices.
`helpers`	Helper functions for generating networks.
`papers`	Methods for generating networks in which papers are vertices.
`terms`	Methods for building networks from terms in bibliographic records.
`topics`	Build networks from topics in a topic model.

Modules

authors Module

Methods for generating networks in which authors are vertices.

`author_cocitation`(papers[, threshold])	Generates an author co-citation network; edges indicate co-citation of authors'
`author_coinstitution`(Papers[, threshold])	Generate a co-institution graph, where edges indicate shared affiliation.
`author_institution`(Papers[, edge_attribs])	Generate a bi-partite graph connecting authors and their institutions.
`author_papers`(papers[, node_id, paper_attribs])	Generate an author_papers network NetworkX directed graph.
`coauthors`(papers[, threshold, edge_attribs])	Generate a co-author network.

Methods

`tethne.networks.authors.`**`author_cocitation`**(*papers*, *threshold=1*, ***kwargs*)

Generates an author co-citation network; edges indicate co-citation of authors' papers.

Similar to papers.cocitation(), except that vertices are authors rather than papers. To generate an author co-citation network, use the networks.authors.author_cocitation() method:

```
>>> ACC = nt.authors.author_cocitation(papers)
>>> ACC
<networkx.classes.graph.Graph object at 0x106571190>
```

Element	Description
Nodes	Author name.
Edge	(a, b) if a and b are referenced by the same paper in papers
Edge attribute	'weight', the number of papers that co-cite a and b.

Parameters **papers** : list

a list of Paper objects.

threshold : int

Minimum number of co-citations required to create an edge between authors.

Returns **cocitation** : networkx.Graph

A cocitation network.

tethne.networks.authors.**author_coinstitution**(*Papers*, *threshold=1*, ***kwargs*)

Generate a co-institution graph, where edges indicate shared affiliation.

Some bibliographic datasets, including data from the Web of Science, includes the institutional affiliations of authors. In a co-institution graph, two authors (vertices) have an edge between them if they share an institutional affiliation in the dataset. Note that data about institutional affiliations varies in the WoS database so this will yield more reliable results for more recent publications.

To generate a co-institution network, use the networks.authors.author_coinstitution() method:

```
>>> ACI = nt.authors.author_coinstitution(papers)
>>> ACI
<networkx.classes.graph.Graph object at 0x106571190>
```

Element	Description
Node	Authors.
Node Attribute	type (string). 'author' or 'institution'.
Edges	(a, b) where a and b are affiliated with the same institution.
Edge attribute	overlap (int). number of shared institutions.

Parameters **Papers** : list

A list of wos_objects.

threshold : int

Minimum institutional overlap required for an edge.

Returns **coinstitution** : NetworkX graph

A coinstitution network.

tethne.networks.authors.**author_institution**(*Papers*, *edge_attribs=[]*, ***kwargs*)

Generate a bi-partite graph connecting authors and their institutions.

This may be slightly ambiguous for WoS data where there is no explicit author-institution mapping. Edge weights are the number of co-associations between an author and an institution, which should help resolve this ambiguity (the more data the better).

Element	Description
Node	Author name.
Edge	(a,b) in E(G) if a and b are authors on the same paper.

Parameters **Papers** : list

A list of `Paper` instances.

edge_attribs : list

List of edge_attributes specifying which `Paper` keys (from the authored paper) to use as edge attributes. For example, the 'date' key in `Paper` .

Returns **author_institution_graph** : networkx.MultiGraph

A graph describing institutional affiliations of authors in the corpus.

`tethne.networks.authors.`**`author_papers`**(*papers*, *node_id='ayjid'*, *paper_attribs=[]*, ***kwargs*)

Generate an author_papers network NetworkX directed graph.

Element	Description
Node	Two kinds of nodes with distinguishing "type" attributes: * type = paper - a paper in papers * type = person - a person in papers Papers node attributes defined by paper_attribs.
Edge	Directed, Author -> his/her Paper.

Parameters **papers** : list

A list of wos_objects.

node_id : string

A key from `Paper` used to identify the nodes.

paper_attribs : list

List of user-provided optional arguments apart from the provided positional arguments.

Returns **author_papers_graph** : networkx.DiGraph

A DiGraph 'author_papers_graph'.

Raises **KeyError** : Raised when node_id is not present in Papers.

`tethne.networks.authors.`**`coauthors`**(*papers, threshold=1, edge_attribs=['ayjid'], **kwargs*)

Generate a co-author network.

As the name suggests, edges are drawn between two author-vertices in the case that those authors published a paper together. Co-authorship networks are popular models for studying patterns of collaboration in scientific communities.

To generate a co-authorship network, use the `networks.authors.coauthors()` method:

Author institutional affiliation is included as a node attribute, if possible.

```
>>> CA = nt.authors.coauthors(papers)
>>> CA
<networkx.classes.multigraph.MultiGraph object at 0x101705650>
```

Element	Description
Node	Author name.
Edges	(a,b) in E(G) if a and b are coauthors on the same paper.

Parameters **papers** : list

A list of `Paper` instances.

threshold : int

Minimum number of co-citations required for an edge. (default: 1)

edge_attribs : list

List of edge_attributes specifying which `Paper` keys (from the co-authored paper) to use as edge attributes. (default: ['ayjid'])

Returns **G** : networkx.Graph

A co-authorship network.

`helpers` Module

Helper functions for generating networks.

`citation_count`(papers[, key, verbose])	Generates citation counts for all of the papers cited by papers.
`simplify_multigraph`(multigraph[, time])	Simplifies a graph by condensing multiple edges between the same node pair into a
`top_cited`(papers[, topn, verbose])	Generates a list of the topn (or topn%) most cited papers.
`top_parents`(papers[, topn, verbose])	Returns a list of `Paper` that cite the topn most cited papers.

`tethne.networks.helpers.`**`citation_count`**(*papers*, *key='ayjid'*, *verbose=False*)

Generates citation counts for all of the papers cited by papers.

Parameters **papers** : list

A list of `Paper` instances.

key : str

Property to use as node key. Default is 'ayjid' (recommended).

verbose : bool

If True, prints status messages.

Returns **counts** : dict

Citation counts for all papers cited by papers.

`tethne.networks.helpers.`**`simplify_multigraph`**(*multigraph*, *time=False*)

Simplifies a graph by condensing multiple edges between the same node pair into a single edge, with a weight attribute equal to the number of edges.

Parameters **graph** : networkx.MultiGraph

E.g. a coauthorship graph.

time : bool

If True, will generate 'start' and 'end' attributes for each edge, corresponding to the earliest and latest 'date' values for that edge.

Returns **graph** : networkx.Graph

A NetworkX `graph` .

`tethne.networks.helpers.`**`top_cited`**(*papers*, *topn=20*, *verbose=False*)

Generates a list of the topn (or topn%) most cited papers.

Parameters **papers** : list

A list of `Paper` instances.

topn : int or float {0.-1.}

Number (int) or percentage (float) of top-cited papers to return.

verbose : bool

If True, prints status messages.

Returns **top** : list

A list of 'ayjid' keys for the topn most cited papers.

counts : dict

Citation counts for all papers cited by papers.

tethne.networks.helpers.**top_parents**(*papers*, *topn=20*, *verbose=False*)

Returns a list of `Paper` that cite the topn most cited papers.

Parameters **papers** : list

A list of `Paper` objects.

topn : int or float {0.-1.}

Number (int) or percentage (float) of top-cited papers.

verbose : bool

If True, prints status messages.

Returns **papers** : list

A list of `Paper` objects.

top : list

A list of 'ayjid' keys for the topn most cited papers.

counts : dict

Citation counts for all papers cited by papers.

papers Module

Methods for generating networks in which papers are vertices.

author_coupling(papers[, threshold, ...])	Vertices are papers and edges indicates shared authorship.
bibliographic_coupling(papers[, ...])	Generate a bibliographic coupling network.
cocitation(papers[, threshold, node_id, ...])	Generate a cocitation network.
direct_citation(papers[, node_id, node_attribs])	Create a traditional directed citation network.
topic_coupling(papers[, threshold, node_id])	Two papers are coupled if they both contain a shared topic above threshold.

Methods

tethne.networks.papers.**author_coupling**(*papers*, *threshold=1*, *node_attribs=['date']*, *node_id='ayjid'*, ***kwargs*)

Vertices are papers and edges indicates shared authorship.

Element	Description
Node	Papers, represented by node_id.
Edge	(a,b) in E(G) if a and b share x authors and x >= threshold
Edge Attributes	overlap: the value of x (above).

Parameters **papers** : list

A list of `Paper`

threshold : int

Minimum number of co-citations required to draw an edge between two authors.

node_id : string

Field in `Paper` used to identify nodes.

node_attribs : list

List of fields in `Paper` to include as node attributes in graph.

Returns **acoupling** : networkx.Graph

An author-coupling network.

`tethne.networks.papers.`**`bibliographic_coupling`**(*papers, citation_id='ayjid', threshold=1, node_id='ayjid', node_attribs=['date'], weighted=False, **kwargs*)

Generate a bibliographic coupling network.

Two papers are **bibliographically coupled** when they both cite the same, third, paper. You can generate a bibliographic coupling network using the `networks.papers.bibliographic_coupling()` method.

```
>>> BC = nt.papers.bibliographic_coupling(papers)
>>> BC
<networkx.classes.graph.Graph object at 0x102eec710>
```

Especially when working with large datasets, or disciplinarily narrow literatures, it is usually helpful to set a minimum number of shared citations required for two papers to be coupled. You can do this by setting the **'threshold'** parameter.

```
>>> BC = nt.papers.bibliographic_coupling(papers, threshold=1)
>>> len(BC.edges())
1216
>>> BC = nt.papers.bibliographic_coupling(papers, threshold=2)
>>> len(BC.edges())
542
```

Element	Description
Node	Papers represented by node_id.
Node Attributes	node_attribs in `Paper`
Edge	(a,b) in E(G) if a and b share x citations where x >= threshold.
Edge Attributes	overlap: the number of citations shared

Parameters **papers** : list

A list of wos_objects.

citation_id: string :

A key from `Paper` to identify the citation overlaps. Default is 'ayjid'.

threshold : int

Minimum number of shared citations to consider two papers "coupled".

node_id : string

Field in `Paper` used to identify the nodes. Default is 'ayjid'.

node_attribs : list

List of fields in `Paper` to include as node attributes in graph.

weighted : bool

If True, edge attribute *overlap* is a float in {0-1} calculated as $\frac{N_{ij}}{\sqrt{N_i N_j}}$ where N_i and N_j are the number of references in `Paper` *i* and *j*, respectively, and N_{ij} is the number of references shared by papers *i* and *j*.

Returns **bcoupling** : networkx.Graph

A bibliographic coupling network.

Raises **KeyError** : Raised when citation_id is not present in the meta_list.

Notes

Lists cannot be attributes? causing errors for both gexf and graphml also nodes cannot be none.

`tethne.networks.papers.`**`cocitation`**(*papers, threshold=1, node_id='ayjid', topn=None, verbose=False, node_attribs=['date'], **kwargs*)

Generate a cocitation network.

A **cocitation network** is a network in which vertices are papers, and edges indicate that two papers were cited by the same third paper. CiteSpace is a popular desktop application for co-citation analysis, and you can read about the theory behind it here. Co-citation analysis is generally performed with a temporal component, so building a `GraphCollection` from a :class'.DataCollection' sliced by `date` is recommended.

You can generate a co-citation network using the `networks.papers.cocitation()` method:

```
>>> CC = nt.papers.cocitation(papers)
>>> CC
<networkx.classes.graph.Graph object at 0x102eec790>
```

For large datasets, you may wish to set a minimum number of co-citations required for an edge between two papers Keep in mind that all of the references in a single paper are co-cited once, so a threshold of at least 2 is prudent. Note the dramatic decrease in the number of edges when the threshold is changed from 2 to 3.

```
>>> CC = nt.papers.cocitation(papers, threshold=2)
>>> len(CC.edges())
8889
>>> CC = nt.papers.cocitation(papers, threshold=3)
>>> len(CC.edges())
1493
```

Element	Description
Node	Cited papers represented by `Paper` ayjid.
Edge	(a, b) if a and b are cited by the same paper.
Edge Attributes	weight: number of times two papers are co-cited together.

Parameters **papers** : list

a list of `Paper` objects.

threshold : int

Minimum number of co-citations required to create an edge.

topn : int or float, or None

If provided, only the topn (int) or topn percent (float) most cited papers will be included in the cocitation network. If None (default), network will include all cited papers (NOTE: this can cause severe memory consumption for even moderately-sized datasets).

verbose : bool

If True, prints status messages.

Returns **cocitation** : networkx.Graph

A cocitation network.

tethne.networks.papers.**direct_citation**(*papers, node_id='ayjid', node_attribs=['date'], **kwargs*)

Create a traditional directed citation network.

Direct-citation graphs are directed acyclic graphs in which vertices are papers, and each (directed) edge represents a citation of the target paper by the source paper. The networks.papers.direct_citation() method generates both a global citation graph, which includes all cited and citing papers, and an internal citation graph that describes only citations among papers in the original dataset.

To generate direct-citation graphs, use the networks.papers.direct_citation() method. Note the size difference between the global and internal citation graphs.

```
>>> gDC, iDC = nt.papers.direct_citation(papers)
>>> len(gDC)
5998
>>> len(iDC)
163
```

Element	Description
Node	Papers, represented by node_id.
Edge	From a paper to a cited reference.
Edge Attribute	Publication date of the citing paper.

Parameters **papers** : list

A list of Paper instances.

node_id : int

A key from Paper to identify the nodes. Default is 'ayjid'.

node_attribs : list

List of user provided optional arguments apart from the provided positional arguments.

Returns **citation_network** : networkx.DiGraph

Global citation network (all citations).

citation_network_internal : networkx.DiGraph

Internal citation network where only the papers in the list are nodes in the network.

Raises **KeyError** : If node_id is not present in the meta_list.

tethne.networks.papers.**topic_coupling**(*papers*, *threshold=0.7*, *node_id='ayjid'*, ***kwargs*)

Two papers are coupled if they both contain a shared topic above threshold.

Element	Description
Node	Papers, represented by node_id.
Edge	(a,b) in E(G) if a and b share >= 1 topics with proportion >= threshold in both a and b.
Edge Attributes	weight: combined mean proportion of each shared topic. topics: list of shared topics.

Parameters **papers** : list

A list of `Paper`

threshold : float

Minimum representation of a topic in each paper.

node_id : string

Field in `Paper` used to identify nodes.

Returns **tc** : networkx.Graph

A topic-coupling network.

terms Module

Methods for building networks from terms in bibliographic records. This includes keywords, abstract terms, etc.

keyword_cooccurrence(papers, threshold[, ...])	Generates a keyword cooccurrence network.
topic_coupling(model[, threshold])	Creates a network of words connected by implication in a common topic(s).

tethne.networks.terms.**keyword_cooccurrence**(*papers*, *threshold*, *connected=False*, ***kwargs*)

Generates a keyword cooccurrence network.

Parameters **papers** : list

A list of `Paper` objects.

threshold : int

Minimum number of occurrences for a keyword pair to appear in graph.

connected : bool

If True, returns only the largest connected component.

Returns **k_coccurrence** : networkx.Graph

A keyword coccurrence network.

tethne.networks.terms.**topic_coupling**(*model*, *threshold=0.005*, ***kwargs*)

Creates a network of words connected by implication in a common topic(s).

Parameters **model** : `LDAModel`

threshold : float

Minimum P(W|T) for coupling.

Returns **tc** : networkx.Graph

A topic-coupling graph, where nodes are terms.

topics Module

Build networks from topics in a topic model.

tethne.networks.topics.**paper_coupling**(*model*, *threshold=0.1*)

tethne.networks.topics.**term_coupling**(*model*, *threshold=0.01*)

readers Package

readers Package

Methods for parsing bibliographic datasets.

dfr	Methods for parsing JSTOR Data-for-Research datasets.
mallet	Reader for output from topic modeling with MALLET.
pubmed	Methods for working with PubMed data are still under development. Please use
wos	Reader for Web of Science field-tagged bibliographic data.

Each file reader provides methods to parse bibliographic data from a scholarly database (e.g. Web of Science or PubMed), resulting in a list of Paper instances containing as many as possible of the following keys (missing values are set to None):

Field	Type	Description
aulast	list	Authors' surnames, as a list.
auinit	list	Authors' initials, as a list.
institution	dict	Institutions with which the authors are affiliated.
atitle	str	Article title.
jtitle	str	Journal title or abbreviated title.
volume	str	Journal volume number.
issue	str	Journal issue number.
spage	str	Starting page of article in journal.
epage	str	Ending page of article in journal.
date	int	Date of publication.
abstract	str	

These keys are associated with the meta data entries in the databases of organizations such as the International DOI Foundation and its Registration Agencies such as CrossRef and DataCite.

In addition, Paper instances will contain keys with information relevant to the networks of interest for Tethne including:

Field	Type	Description
citations	list	List of minimum Paper instances for cited references.
ayjid	str	First author's name (last, fi), publication year, and journal.
doi	str	Digital Object Identifier.
pmid	str	PubMed ID.
wosid	str	Web of Science UT fieldtag.

Missing data here also results in the above keys being set to None.

exception tethne.readers.**DataError**(*value*)

Bases: exceptions.Exception

tethne.readers.**merge**(*P1, P2, fields=['ayjid']*)

Combines two lists (P1 and P2) of `Paper` instances into a single list, and attempts to merge papers with matching fields. Where there are conflicts, values from `Paper` in P1 will be preferred.

Parameters **P1** : list

A list of `Paper` instances.

P2 : list

A list of `Paper` instances.

fields : list

Fields used to identify matching `Paper`

Returns **combined** : list

A list of `Paper` instances.

Examples

```
>>> import tethne.readers as rd
>>> P1 = rd.wos.read("/Path/to/data1.txt")
>>> P2 = rd.dfr.read("/Path/to/DfR")
>>> papers = rd.merge(P1, P2, ['ayjid'])
```

`dfr` Module

Methods for parsing JSTOR Data-for-Research datasets.

`ngrams`(datapath[, N, ignore_hash, ...])	Yields N-grams from a JSTOR DfR dataset.
`read`(datapath)	Yields `Paper` s from JSTOR DfR package.

tethne.readers.dfr.**from_dir**(*path*)

Convenience function for generating a list of `Paper` from a directory of JSTOR DfR datasets.

Parameters **path** : string

Path to directory containing DfR dataset directories.

Returns **papers** : list

A list of `Paper` objects.

Raises **IOError** :

Invalid path.

Examples

```
>>> import tethne.readers as rd
>>> papers = rd.dfr.from_dir("/Path/to/datadir")
```

tethne.readers.dfr.**ngrams**(*datapath*, *N='bi'*, *ignore_hash=True*, *apply_stoplist=False*)

Yields N-grams from a JSTOR DfR dataset.

Parameters **filepath** : string

Filepath to unzipped JSTOR DfR folder containing N-grams (e.g. 'bigrams').

N : string

'bi', 'tri', or 'quad'

ignore_hash : bool

If True, will exclude all N-grams that contain the hash '#' character.

apply_stoplist : bool

If True, will exclude all N-grams that contain words in the NLTK stoplist.

Returns **ngrams** : dict

Keys are paper DOIs, values are lists of (Ngram, frequency) tuples.

Examples

```
>>> import tethne.readers as rd
>>> trigrams = rd.dfr.ngrams("/Path/to/DfR", N='tri')
```

tethne.readers.dfr.**read**(*datapath*)

Yields `Paper` s from JSTOR DfR package.

Each `Paper` is tagged with an accession id for this read/conversion.

Parameters **filepath** : string

Filepath to unzipped JSTOR DfR folder containing a citations.XML file.

Returns **papers** : list

A list of `Paper` objects.

Examples

```
>>> import tethne.readers as rd
>>> papers = rd.dfr.read("/Path/to/DfR")
```

mallet Module

Reader for output from topic modeling with MALLET.

tethne.readers.mallet.**load**(*top_doc*, *word_top*, *topic_keys*, *Z*, *metadata=None*, *metadata_key='doi'*)

Parse results from LDA modeling with MALLET.

MALLET's LDA topic modeling algorithm produces a collection of output files. `read()` takes the topic-document and (sparse) word-topic matrices, as tab-separated value files, along with a metadata file that maps each MALLET document id to a `Paper`, using the *metadata_key*.

Parameters **top_doc** : string

Path to topic-document datafile generated with –output-doc-topics.

word_top : string

Path to word-topic datafile generated with –word-topic-counts-file.

topic_keys : string

Path to topic-keys datafile generated with –output-topic-keys.

Z : int

Number of topics.

metadata : string (optional)

Path to tab-separated metadata file with IDs and `Paper` keys.

Returns **ldamodel** : `LDAModel`

`tethne.readers.mallet.`**`read`**(*top_doc*, *word_top*, *topic_keys*, *Z*, *metadata=None*, *metadata_key='doi'*)

Generates `Paper` objects from Mallet output.

Each `Paper` is assigned a topic vector.

Parameters **top_doc** : string

Path to topic-document datafile generated with –output-doc-topics.

word_top : string

Path to word-topic datafile generated with –word-topic-counts-file.

topic_keys : string

Path to topic-keys datafile generated with –output-topic-keys.

Z : int

Number of topics.

metadata : string (optional)

Path to tab-separated metadata file with IDs and `Paper` keys.

Returns **papers** : list

List of `Paper`

`pubmed` Module

Methods for working with PubMed data are still under development. Please use with care.

`read`(filepath)	Given a file with PubMed XML, return a list of `Paper` instances.

`tethne.readers.pubmed.`**`read`**(*filepath*)

Given a file with PubMed XML, return a list of `Paper` instances.

See the following hyperlinks regarding possible structures of XML: * http://www.ncbi.nlm.nih.gov/pmc/pmcdoc/tagging-guidelines/citations/v2/citationtags.html#2Articlewithmorethan10authors%28
* http://dtd.nlm.nih.gov/publishing/

Each `Paper` is tagged with an accession id for this read/conversion.

Usage

```
>>> import tethne.readers as rd
>>> papers = rd.pubmed.read("/Path/to/PubMedData.xml")
```

Parameters **filepath** : string

Path to PubMed XML file.

Returns **meta_list** : list

A list of `Paper` instances.

`wos` Module

Reader for Web of Science field-tagged bibliographic data.

Tethne parses Web of Science field-tagged data into a list of `Paper` objects. This is a two-step process: data are first parsed into a list of dictionaries with field-tags as keys, and then each dictionary is converted to a `Paper` . `readers.wos.read()` performs both steps in sequence.

One-step Parsing The method `readers.wos.read()` performs both `readers.wos.parse()` and `readers.wos.convert()` . This is the preferred (simplest) approach in most cases.

```
>>> papers = rd.wos.read("/Path/to/savedrecs.txt")
>>> papers[0]
<tethne.data.Paper instance at 0x101b575a8>
```

Alternatively, if you have many data files saved in the same directory, you can use `readers.wos.from_dir()` :

```
>>> papers = rd.wos.parse_from_dir("/Path/to")
```

Two-step Parsing Use the two-step approach if you need to access fields not included in `Paper`, or if you wish to perform some intermediate manipulation on the raw parsed data.

First import the `readers.wos` module:

```
>>> import tethne.readers as rd
```

Then parse the WoS data to a list of field-tagged dictionaries using `readers.wos.parse()` :

```
>>> wos_list = rd.wos.parse("/Path/to/savedrecs.txt")
>>> wos_list[0].keys()
['EM', '', 'CL', 'AB', 'WC', 'GA', 'DI', 'IS', 'DE', 'VL', 'CY', 'AU', 'JI',
 'AF', 'CR', 'DT', 'TC', 'EP', 'CT', 'PG', 'PU', 'PI', 'RP', 'J9', 'PT',
 'LA', 'UT', 'PY', 'ID', 'SI', 'PA', 'SO', 'Z9', 'PD', 'TI', 'SC', 'BP',
 'C1', 'NR', 'RI', 'ER', 'SN']
```

Convert those field-tagged dictionaries to `Paper` objects using `readers.wos.convert()` :

```
>>> papers = rd.wos.convert(wos_list)
>>> papers[0]
<tethne.data.Paper instance at 0x101b575a8>
```

`convert`(wos_data)	Convert parsed field-tagged data to `Paper` instances.
`from_dir`(path)	Convenience function for generating a list of `Paper` from a
`parse`(filepath)	Parse Web of Science field-tagged data.
`read`(datapath)	Yields a list of `Paper` instances from a Web of Science data file.

Methods

exception tethne.readers.wos.**DataError**
Bases: exceptions.Exception

tethne.readers.wos.**convert**(*wos_data*)
Convert parsed field-tagged data to Paper instances.

Convert a dictionary or list of dictionaries with keys from the Web of Science field tags into a Paper instance or list of Paper instances, the standard for Tethne.

Each Paper is tagged with an accession id for this conversion.

Parameters **wos_data** : list

A list of dictionaries with keys from the WoS field tags.

Returns **papers** : list

A list of Paper instances.

Notes

Need to handle author name anomolies (case, blank spaces, etc.) that may make the same author appear to be two different authors in Networkx; this is important for any graph with authors as nodes.

Examples

```
>>> import tethne.readers as rd
>>> wos_list = rd.wos.parse("/Path/to/data.txt")
>>> papers = rd.wos.convert(wos_list)
```

tethne.readers.wos.**from_dir**(*path*)
Convenience function for generating a list of Paper from a directory of Web of Science field-tagged data files.

Parameters **path** : string

Path to directory of field-tagged data files.

Returns **papers** : list

A list of Paper objects.

Raises **IOError** :

Invalid path.

Examples

```
>>> import tethne.readers as rd
>>> papers = rd.wos.from_dir("/Path/to/datadir")
```

tethne.readers.wos.**parse**(*filepath*)
Parse Web of Science field-tagged data.

Parameters **filepath** : string

Filepath to the Web of Science plain text file.

Returns **wos_list** : list

A list of dictionaries each associated with a paper from the Web of Science with keys from docs/fieldtags.txt as encountered in the file; most values associated with keys are strings with special exceptions defined by the list_keys and int_keys variables.

Raises **KeyError** : Key value which needs to be converted to an 'int' is not present.

AttributeError : :

IOError : File at filepath not found, not readable, or empty.

Notes

Unknown keys: RI, OI, Z9

Examples

```
>>> import tethne.readers as rd
>>> wos_list = rd.wos.parse("/Path/to/data.txt")
```

tethne.readers.wos.**read**(*datapath*)

Yields a list of `Paper` instances from a Web of Science data file.

Parameters **datapath** : string

Filepath to the Web of Science field-tagged data file.

Returns **papers** : list

A list of `Paper` instances.

Examples

```
>>> import tethne.readers as rd
>>> papers = rd.wos.read("/Path/to/data.txt")
```

utilities Package

utilities Package

Helper functions for `tethne.networks` .

class tethne.utilities.**Dictionary**

A two-way index for integer/string pairs.

tethne.utilities.**attribs_to_string**(*attrib_dict*, *keys*)

A more specific version of the subdict utility aimed at handling node and edge attribute dictionaries for NetworkX file formats such as gexf (which does not allow attributes to have a list type) by making them writable in those formats

tethne.utilities.**concat_list**(*listA*, *listB*, *delim=' '*)

Concatenate list elements pair-wise with the delim character Returns the concatenated list Raises index error if lists are not parallel

tethne.utilities.**contains**(*l*, *f*)

Searches list l for a pattern specified in a lambda function f.

tethne.utilities.**dict_from_node**(*node*, *recursive=False*)
Converts ElementTree node to a dictionary.

Parameters **node** : ElementTree node

recursive : boolean

If recursive=False, the value of any field with children will be the number of children.

Returns **dict** : nested dictionary.

Tags as keys and values as values. Sub-elements that occur multiple times in an element are contained in a list.

tethne.utilities.**overlap**(*listA*, *listB*)
Return list of objects shared by listA, listB.

tethne.utilities.**strip_non_ascii**(*string*)
Returns the string without non-ASCII characters.

Parameters **string** : string

A string that may contain non-ASCII characters.

Returns **clean_string** : string

A string that does not contain non-ASCII characters.

tethne.utilities.**subdict**(*super_dict*, *keys*)
Returns a subset of the super_dict with the specified keys.

writers Package

writers Package

Export networks to structured and unstructured formats, for visualization.

collection	Write GraphCollection to a structured data format.
graph	Write NetworkX graphs to structured and unstructured network file formats.
matrix	Methods for writing matrices to commonly-used file formats, for external visualization and analysis.

collection Module

Write GraphCollection to a structured data format.

to_dxgmml(C, path)	Writes a GraphCollection to

tethne.writers.collection.**to_dxgmml**(*C*, *path*)
Writes a GraphCollection to dynamic XGMML..

Dynamic XGMML is a schema for describing dynamic networks in Cytoscape 3.0. This method assumes that *Graph* indices are orderable points in time (e.g. years). The "start" and "end" of each node and edge are determined by periods of consecutive appearance in the GraphCollection . Node and edge attributes are defined for each *Graph*. in the GraphCollection.

For example, to build and visualize an evolving co-citation network:

```
>>> # Load some data.
>>> import tethne.readers as rd
>>> papers = rd.wos.read(datapath)

>>> # Build a DataCollection, and slice it temporally using a
>>> #  4-year sliding time-window.
>>> from tethne.data import DataCollection, GraphCollection
>>> D = DataCollection(papers)
>>> D.slice('date', 'time_window', window_size=4)

>>> # Generate a GraphCollection of co-citation graphs.
>>> from tethne.builders import paperCollectionBuilder
>>> builder = paperCollectionBuilder(D)
>>> C = builder.build('date', 'cocitation', threshold=2)

>>> # Write the GraphCollection as a dynamic network.
>>> import tethne.writers as wr
>>> wr.collection.to_dxgmml(C, "/path/to/network.xgmml")
```

Parameters **C** : GraphCollection

The GraphCollection to be written to XGMML.

path : str

Path to file to be written. Will be created/overwritten.

Notes

Period start and end dates in this method are inclusive, whereas XGMML end dates are exclusive. Hence +1 is added to all end dates when writing XGMML.

graph Module

Write NetworkX graphs to structured and unstructured network file formats.

Many methods simply invoke equivalent methods in NetworkX.

to_gexf(graph, output_path)	Writes graph to GEXF.
to_graphml(graph, output_path)	Writes graph to GraphML.
to_sif(graph, output_path)	Generates Simple Interaction Format output file from provided graph.

tethne.writers.graph.**to_gexf**(*graph*, *output_path*)

Writes graph to GEXF.

Uses the NetworkX method write_gexf.

Parameters **graph** : networkx.Graph

The Graph to be exported to GEXF.

output_path : str

Full path, including filename (without suffix). e.g. using "./graphFolder/graphFile" will result in a GEXF file at ./graphFolder/graphFile.gexf.

tethne.writers.graph.**to_graphml**(*graph*, *output_path*)
Writes graph to GraphML.

Uses the NetworkX method write_graphml.

Parameters **graph** : networkx.Graph

The Graph to be exported to GraphML.

output_path : str

Full path, including filename (without suffix). e.g. using "./graphFolder/graphFile" will result in a GraphML file at ./graphFolder/graphFile.graphml.

tethne.writers.graph.**to_sif**(*graph*, *output_path*)
Generates Simple Interaction Format output file from provided graph.

The SIF specification is described here.

to_sif() will generate a .sif file describing the network, and a few .eda and .noa files containing edge and node attributes, respectively. These are equivalent to tab-delimited tables, and can be imported as such in Cytoscape 3.0.

Parameters **graph** : networkx.Graph

The Graph to be exported to SIF.

output_path : str

Full path, including filename (without suffix). e.g. using "./graphFolder/graphFile" will result in a SIF file at ./graphFolder/graphFile.sif, and corresponding .eda and .noa files.

tethne.writers.graph.**to_table**(*graph*, *path*)

matrix Module

Methods for writing matrices to commonly-used file formats, for external visualization and analysis. Not yet implemented.

CHAPTER

TWO

ABOUT

Tethne is developed by the ASU Digital Innovation Group (DigInG), part of the Laubichler Lab in the Center for Biology & Society, School of Life Sciences.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2011131209, and NSF Doctoral Dissertation Research Improvement Grant No. 1256752.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

t

PYTHON MODULE INDEX

t