Netshape Documentation

Release 0.1

Matthew Smith

October 21, 2016

Contents

`netshape` is a versatile command-line tool designed to build interactive, browser-based, and deployable visualizations of networks. This enables a more collaborative and more informative visualization process.

With `netshape`, a network edgelist is turned into a web-ready interactive visualization with just one command, and we can view that visualization with just a few more:

```
> netshape build AwesomeNetwork.csv
> cd dist
> netshape server
> echo "let the visualization begin"
```

Netshape is also highly configurable and hackable; it supports custom commands, which enables the development of complex visualization pipelines.

CHAPTER 1

Key Features

Web-ready visualization: Building and sharing visualizations of networks can help convey complex or technical properties of a network to non-technical users. `netshape` enables the instantanious sharing of these visualizations.

Flexible configuration: `netshape` commands are configured through a `netshape_conf.py` file, which is *just a python file*. This means that any normal python code can be included or imported in you visualization pipeline.

CHAPTER 2

Dependencies

Before installing, ensure that python is installed on your computer, and is at least version 2.7.

CHAPTER 3

Installation

Once the dependencies have been installed, in order to install `netshape`, simply run:

```
> git clone https://github.com/m-smith/netshape
> cd netshape
> make install
```

CHAPTER 4

Contributing / Testing

Some rudimentary tests have been written, though many more could be made. Any contribution to this project is highly appreciated.

In order to run the existing tests, just use: `make test` from the project root

4.1 Documentation

4.1.1 Introduction

`netshape` is a command-line that is designed to help scaffold browser-based network visualizations, so that they can be shared with others in an interactive manner while still remaining lightweight. Often it is useful to zoom and pan around a network in order to get a sense of particular details that any static image could not capture. `netshape` is the solution to that problem.

Installation

See: Overview for installation instructions.

Default Usage

By default, `netshape` offers two subcommands, though it can be easily extended to include more.

`netshape build` accepts as arguments the name of a csv-formatted edgelist file (JSON network option coming soon), or a csv-formatted edgelist from `stdin`, as well as some optional parameters. The eigenvector centrality, degree, and a modularity maximizing community assignment is then computed for all nodes. A json representation of this network is then used alongside the d3.js library to build a visualization of the network in the browser, by default in a subfolder named 'dist' which can be coloured according to the statistics that were computed earlier.

Arguments

`network`
The csv formatted edgelist representing the network to be visualized. Also accepts from `stdin`

`-h, --help`
Displays a help message.

`-s` `SEP`**`, --seps`** `SEP`
the string delimiter between values in the edgelist (default: ",")

-d DIRECTED, **--directed** DIRECTED
Boolean indicator - true if the graph is directed (default: True)

-o OUT, **--out** OUT
The name of the file to build the web visualization in (default: "dist")

-n NAME, **--name** NAME
The name of the visualization in that appears in the browser UI (default: "Network")

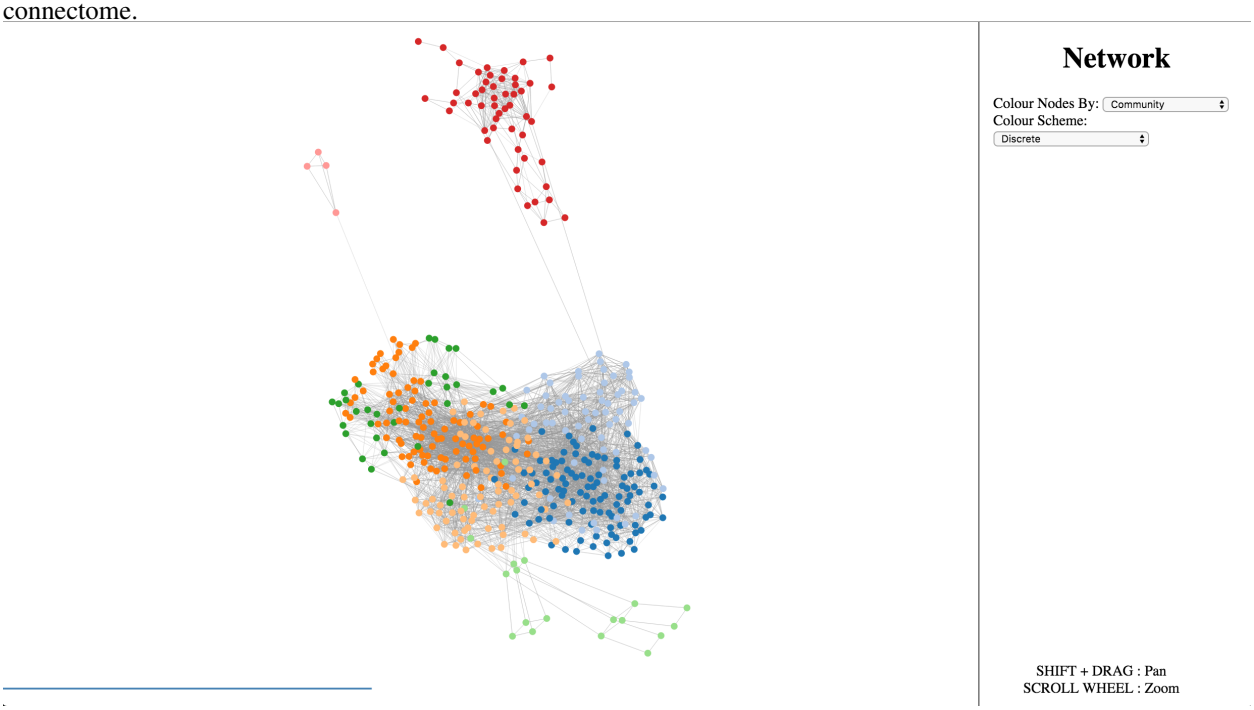
netshape serve a utility function that serves the web visualization on a localhost server (port 8000 by default)

Arguments

-p PORT, **--port** PORT
The port to serve on.

Web Interface

Below is a sample view of the default web interface for the visualization. The particular network is a *C. Elegans* neural connectome.



There are options on the right side to colour the network according to the various network statistics that were computed before it was built, as well as choose between discrete and continuous colour schemes. It is also possible to zoom and pan the view of the network. When the mouse hovers over a node, that node's ID will appear above the node. Nodes are arranged according to a force-directed layout, but can be dragged interactively by clicking and dragging with the mouse.

4.1.2 Configuration

In addition to the previously mentioned options, `netshape` will optionally and automatically load a `netshape_conf.py` file in the directory where netshape is being run. This file allows a user to write custom functions for computing network properties, add additional subcommands, or modify existing ones.

The `netshape_conf.py` file

The `netshape_conf.py` is simply a python module that is imported before the command line arguments are parsed. This allows this module to define additional command line arguments or change the meaning of existing ones before they are used to build the network visualization.

The `conf` object

Every `netshape_conf.py` imports a global `conf` object from the `netshape.config` module that is shared across the `netshape` application. `conf` object maintains and exposes the configuration state using an internal dictionary of `Command` objects, each of which represents a single subcommand of the application. This dictionary can be accessed as the `commands` property of the `conf` object. `Command`'s are indexed by the name of the command itself (e.g. for `netshape build`, it would be `conf.commands["build"]`).

The `conf` object also exposes a method to add commands to the application. This could be useful if a user wanted to make different visualizations of the same network, or required other side effects, like outputting node properties in csv format. This function is accessed via `conf.register_command()`, and accepts a required `command_name` argument, as well as optional `help` which is a string representing the help message for that subcommand, and `command_args`, a list of dicts of arguments to be passed to the `argparse` module's `add_argument` function, as well as the additional 'name' key, which specifies the list of possible names for that parameter.

This enables us to build custom command line arguments which can be used gracefully in the network visualization pipeline.

The `Command` class

The command class allows us to construct and modify visualization pipelines that serve a common purpose. Currently it is somewhat underdeveloped, and as such, only exposes two methods: `Command.read_edges()`, which builds builds the original network or adds edges to an existing one. It accepts a single parameter, which is the name of the command line argument that was used to capture the edgelist file. The other method, :code:'Command.add_node_prop'() can be used to add a property to the network. It accepts a name for the property as it's first argument, a function as it's second argument, which has some restrictions on it's arguments, and, additional args and kwargs to pass to that function when it is being called. The function must receive a netshape object as it's first argument, the name of the property being added as the second, and a dictionary of command line arguments as the third argument. This function then can compute a network statistic, and call,

> `Netshape.add_node_prop()` to add it to the netshape object. These methods are added in order, and executed sequentially to compute all desired network statistics. Then, internal methods are used to build the directory for the web page.

A sample `netshape_conf.py`

In order to reduce the potential confusion of the above examples, we give an example of a `netshape_conf.py` file which would result in the default commands. The `node_props` module adds utility functions that compute various node properties of the network, according to the function format described above. For more details on specific functions, see the Full API

```
import argparse
import sys

from netshape.config import conf
from netshape import node_props
```

```
(conf
 .register_command('build', help="build a static visualization of a network",
                   command_args=build_args)
 .read_edges('network')
 .add_node_prop('Eigenvector Centrality', node_props.eigenvector_centrality)
 .add_node_prop('Community', node_props.modularity_community)
 .add_node_prop('Degree', node_props.degree)
)
(conf
 .register_command('serve', help="utility function to view a network visualization", command_args=[{
        "name": ['-p', '--port'],
        "dest": 'p',
        "default":8000,
        "help": "The port to serve on."
  }])
 .add_node_prop('', serve)

 build_args = [{
               'name': ['network'],
               'nargs': "?",
               'help':"the network, formatted as a csv edgelist - "
                      "accepts a path or from stdin",
               "type":argparse.FileType('r'),
               "default":sys.stdin
           }, {
               "name": ['-s', '--seps'],
               "dest": 'sep',
               "default":",",
               "help": "the string delimiter between values in the edgelist"
                       " (default: \",\")"
           }, {
               "name": ['-d', '--directed'],
               "dest": 'directed',
               "default":"True",
               "help": "Boolean indicator that is True if the network is directed, "
                       "and False otherwise (default: True)"
           }, {
               "name": ['-o', '--out'],
               "dest": 'out',
               "default":"dist",
               "help": "The name of the file to build the web visualization in "
                       "(default: \"dist\")"
           }, {
               "name": ['-n', '--name'],
               "dest": 'name',
               "default":"Network",
               "help": "The name of the file to build the web visualization in "
                       "(default: \"Network\")"
           }]
```

4.1.3 Python API

netshape.main

The main netshape module which contains the Netshape graph representation class

class netshape.main.**Netshape**

The netshape class. Netshape objects maintain the network state through two primary attributes:

self.json_graph The JSON representation of the graph, which maintains nodes and links as lists of dicts. This is what is eventually written to the data file used for the web UI.

```
self.json_graph = {'nodes':  [], 'links':[]}
```

self.g a networkx representation of the network. This is useful for computation of network statistics.

add_node_prop(*prop*, *prop_map*)
Adds a node property to the json_graph *Positional arguments:*

prop
The name of the property to add

prop_map
A mapping from node ID's to property values

build_visualization(*out='dist'*, *title='Network'*)
Scaffolds the visualization project into a directory named dist

from_edgelist(*edgelist*, *sep=','*, *directed=True*)
Reads a csv formatted edgelist and adds nodes and edges to the network accordingly. If there is a header, the columns labeled 'source', and 'target' will be used as node ID's, and any other columns, edge attributes. Otherwise, the first two columns will be used as node ID's.

Positional arguments:

edgelist
the file to be opened

Keyword arguments:

sep
the delimiter string in the csv (default: " , ")

directed
boolean indicating true if the graph is directed (default: True)

netshape.config

Module for writing netshape-config.py files

class netshape.config.**Command**
docstring for Command

add_node_prop(*name*, *func*, **args*, ***kwargs*)
Adds a property to nodes in the network

Positional arguments: name – the name of the property to be added func – a function which accepts a netshape object as the first

argument, the name of the property as the second, and the command line arguments as the third, as well as any additional arguments, and returns a mapping from node ids to property values

run_pipeline(*cl_args*)
Executes the pipeline

class netshape.config.**Config**
docstring for Config

register_command(*command_name*, *help=None*, *command_args=None*)
Registers a subcommand and returns a command object which can

netshape.node_props

netshape.node_props.**eigenvector_centrality**(*ns*, *name*, _)
Computes eigenvector centrality for all nodes

netshape.node_props.**modularity_community**(*ns*, *name*, _)
Community detection

n

Symbols

A

B

C

D

E

F

M

N

P

R

S