The netdiff, a Module to Find Differences Between Complex Networks

Árpád Horváth
Alba Regia University Centre of Óbuda University,
H-8000 Székesfehérvár, Budai út 45., Hungary
Email: horvath.arpad@arek.uni-obuda.hu

***Abstract*—We have developed a module to find the differences between complex networks. Its goal is to find the main properties of the evolution of complex network from snapshots stored in several files. The module can find new, removed and changed nodes and edges in a network pair. We present its working using the more then 2000 saved files of the software package distribution of the Ubuntu distribution.**

I. Introduction

Complex networks are systems that can be described mathematically as a series of graphs. These networks includes usually more thousands or millions of entities, called vertices or nodes. All pairs of these vertices can be connected or not. These connections are called edges. In the case of Internet the vertices are the routers and computers, and the edges are the cables. In the Internet the vertices in the both end of the edges have the same role: both of them can send or receive data. These type of networks are called undirected networks. One example of the other type of networks is the Web, where the web pages are the vertices and the links are the edges. This network is directed: the page having link to the other and the linked page have different roles in the connection.

These networks change: vertices and edges appears and disappears, therefore the another graph describes the network in one moment and another graph in the other. The dynamics of networks have been extensively studied in the last decade[1], [2], [3]. These investigations needs to collect information about these changes. The aim of the netdiff module is to provide easy to use tools for the data requisition.

II. General properties of the netdiff module

The netdiff module have been written in Python language, a general purpose dynamically typed object oriented programming language [4]. The cxnet module is platform independent: it can be used on Windows, on Linux on Mac Os X, and on every operating system, where the Python interpreter is available. It uses the igraph Python module to store and handle the networks[5].

The network module can deals with a collection of network data stored in one of the file formats that igraph can read. These are in fact not different networks, but snapshots of the same network in several moments. The netdiff module have two stages to collect this data. In the first one it collects the properties of the networks and the differences of two successive snapshots of the network and it stores in Python shelves, a binary file format can be handled by Python. This shelves include serialized Python objects.

In the second stage netdiff can collect global properties of the changes:

- What hours or week days are the most changes on?
- Is there preferential attachment in the network? Preferential attachment means, that the new vertices tends to connect vertices with large degree.

III. The details and the usage of the netdiff module

The netdiff module needs at least two networks with the properties below:

- The vertices needs to have `name` argument. This must be unique for every vertex.
- The networks need to store the date in a given string format.

The netdiff module have four main classes.

- The `Extractor` class creates the list of the snapshots of networks to process. It uses the `NetworkProperties` and `NetworkDiff` classes to get the properties of the networks and the differences between the successive snapshots. It saves the differences into a Python shelf named `diffs.shelf` and the properties of the networks into shelves named `<filename>.shelf` where the `<filename>` is the name of the file the networks is stored without the extension.
- The `Summarizer` class is responsible for the second stage. This groups the network changes by the day of the week and the hour of the day and collect the data that need to check whether preferential attachment is present in the network or not.

To create an object from the `Extractor` class one need to give the parameters below:

inputs
: The files we want to process. This can be a list of file names (list of strings) or a pattern for the file names (string) like `"ubuntu*.graphmlz"`. In the letter case the extraction goes through the files in alphabetic order.

whatched_attributes
: The attributes we pay attention to, when we collect the changed vertices and changed

edges. This is a dictionary with two keys: `vertex` and `edge`. The value of the `whatched_properties["vertex"]` is a list of vertex attributes, and `whatched_properties["vertex"]` is a list of edge attributes.

stored_attributes
: The attributes we want to store for new, deleted or changed names. This is a dictionary with the same two keys as in the whatched_properties, but there can be attributes in the lists starting with underscore, that implicate to store a value given by a method of the vertex or edge instead of attribute. If the string `"_degree"` is in the `stored_properties["vertex"]` then the degree of the vertex will be stored whenever a vertex is stored.

time
: The name of the graph attribute, that stores the date and time of the stored snapshot. Its default value is `"time"`.

time_format
: The format of the time attribute. Its default value is `"%Y-%m-%d %H:%M:%S GMT"` so times like `"2012-09-01 00:00:07 GMT"` can be interpret by default.

Calling the `run` method of the Extractor object starts to extract the differences and network properties from the files of the `input` parameter.

The `NetworkProperties` object returns the properties of the network as a data structure easy to store by the `Extractor` object. It does not deals with files, just the `Graph` objects of the `igraph` module. A `NetworkProperties` object can be created with the next parameters:

network
: The `network` object of the `igraph.Graph` class.

time
: The same as in the `Extractor` class.

time_format
: The same as in the `Extractor` class.

The `NetworkDiff` object returns the differences of two networks as a data structure easy to store by the `Extractor` object. We call the first network as reference network, and the second one as network. New vertices or edges are ones, that are absent from the reference network, and present in the network, deleted vertices or edges are the ones that are present in the reference network and absent in the network. A `NetworkDiff` object can be created with the next parameters:

refnet_properties
: A `NetworkProperties` instance created using the reference network.

net_properties
: A `NetworkProperties` instance created using the network.

whatched_attributes
: The same as in the `Extractor` class.

stored_attributes
: The same as in the `Extractor` class.

The netdiff module has two example script to easy made the first and the second stages of the data processing, `extractor.py` and `summarizer.py` respectively. This can be copy and modify for the needs.

The netdiff module was prepared to handle big amount of network date. If we want to break the running with a keyboard interrupt. It closes the files, and at the next run the extraction process will counted where it was finished.

IV. Conclusion

The netdiff module is a big help for finding the dynamic properties of the networks. Our goal is to investigate the more then 2500 snapshots of the software package dependency network of the Ubuntu Linux distribution [6]. This snapshots have been collected from the mid-July of 2012 until now in every hours.

References

[1] H. Jeong, Z. Néda, and A. L. Barabási, "Measuring preferential attachment in evolving networks," *EPL (Europhysics Letters)*, pp. 567+, Feb. 2003. [Online]. Available: http://dx.doi.org/10.1209/epl/i2003-00166-9

[2] M. E. J. Newman, *Networks, An Introduction*. Oxford University Press, 2010.

[3] M. Barthelemy, A. Barrat, R. Pastor-Satorras, and A. Vespignani, "Dynamical patterns of epidemic outbreaks in complex heterogeneous networks," *JOURNAL OF THEORETICAL BIOLOGY*, vol. 235, no. 2, pp. 275–288, JUL 21 2005. [Online]. Available: http://arxiv.org/abs/cond-mat/0410330

[4] M. Summerfield, *Programming in Python 3: A Complete Introduction to the Python Language*. Addison-Wesley Professional, 2010.

[5] G. Csárdi and T. Nepusz, "The igraph software package for complex network research," *InterJournal Complex Systems*, p. 1695, 2006.

[6] A. Horváth, "Studying complex networks with cxnet," *Acta Physica Debrecina*, vol. XLIV, 2010.