# Analyzing small world phenomenon and scale free network of python packages

Avadhoot Agasti
aagasti@indiana.edu

Abhishek Gupta
abhigupt@iu.edu

## ABSTRACT

The *small world* phenomenon is a very important and interesting concept which can be practically used in optimizing imports for programming languages like python, java etc. This will also help us understand the network of the packages used by developers and classify them as scale free or random network. Further, this analysis can be extended to any other programming language and understand the package network. The scope of this project is to focus on python based packages and navigate to all dependent packages to build a network graph of these packages. Further, identify the important packages which could be like important nodes in scale-free network.

## KEYWORDS

scale-free, random networks, small-world, packages

## 1 INTRODUCTION

When try to install a python library or rather when we use this python library first thing that we need to do is we have to *pip install* it. Most of the times, a python library is dependent on another library. Our idea is to build a network or graph of the python library dependency. In this network, every library will be the node of the network and the dependency will form the link. Such a graph can be queried to answer several questions like

- Which are the most core packages which are widely, directly or undirectly, used in large number of other packages?
- In which subject-area the new development is happening. We can pivot this solution around the small number of packages which are included in large number of packages. For example, if networkX is being used by lot of new packages then we can say that there is lot of development happening in network science
- What all packages will be impacted due to changes in a base package (e .g. if we find a severe bug in networkX, what are the other packages which can be potentially impacted due to the bug fix)

## 2 REQUIREMENTS

The goal of this analysis is

- to create a network graph of these dependencies
- understand the dependencies and important nodes
- identify if its a scale-free network
- identify any other properties depicted by this graph
- update the graph and build the graph as and when needed

At the end of this exercise we should be able to run this analysis on available python [1] packages. Also, we need understand and compute the other properties of the graph like average degree, average clustering co-efficient, average path length etc Also verify if the graph is *scale free* or just a *random* graph.

## 3 TECHNICAL SOLUTION

### 3.1 Data Sources

Our approach is to identify the dependent packages by looking at the dependencies using *pip show* output. For example

```
pip show networkx

Name: networkx

Version: 1.11

Summary: Python package for creating and
manipulating graphs and networks

Home-page: http://networkx.github.io/

Author: NetworkX Developers

Author-email: networkx-discuss@googlegroups.com

License: BSD

Location: /Libs/anaconda/lib/python3.6/site-packages

Requires: decorator
```

Looking at the above output we can clearly see that *networkxx* depends on package *decorator* and further package *decorator* may be dependent on other package and so on. We continue to traverse we should be able to find all dependent packages till we reach code python libraries. Interestingly, there *128k* python packages available [1] which should give us a lot of data points for our analysis.

### 3.2 Automation

We can automate running this analysis on these packages using python script. After collecting this information each package will form a node in the graph and each edge will represent the dependency with the next module. We should be able to plot this graph using *networkx* module and represent the most important nodes which become the hubs to the network. Also, compute the degree and clustering co-efficient for this graph. Further, we can also study to graph to identify if the graph follows the *power law* distribution or it is a *scale free* network or its just a *random* network.

### 3.3 Technology

We plan to perform most of our coding in Python itself. We plan to use Gephi for visualizing the network.

### 3.4 Challanges

During this exercise we can run into few technical challenges for example to perform this analysis we have to install each python package locally which could be really resource intensive specially in terms of disk space. Also, while doing the analysis and loading the graph for 128k packages could be memory and cpu intensive. To get around this problem we may have to reduce the analysis to less number of python packages and keep adding mode packages as we make progress.

## 4 RELATED WORK

- The project Pipedtree - [2] is command line utility which allows user to see the installed packages in the form of dependency tree. However, this utility is focused more on solving dependency conflicts than the kind of analysis we propose to perform.
- Analysis of 30K Github projects [3]. The project was aimed to analyze the 30k different Java, Ruby and Javascript projects on Github to understand the top libraries being used. While their analysis was similar to what we plan to do, the approach was not network based.

## 5 ACKNOWLEDGEMENTS

The authors thank Prof. YY Ahn for his technical guidance. The authors would also like to thank TAs of Network Science class for their valued support.

## 6 REPO

All project and report document can be found at github project.

## REFERENCES

[1] Python.org. Python, packages. Web Page. Accessed: 2017-02-02.
[2] Python.org. pipdeptree, command line utility to show dependency tree of packages. Web Page. Accessed: 2017-02-02.
[3] Tal Weiss. OverOps, blog by tal weiss. Web Page. Accessed: 2017-02-02.