Complex networks in the higher education

Árpád Horváth
Budapest Tech, Center for Innovation and Education
H-8000 Székesfehérvár, Budai út 45., Hungary
Email: horvath.arpad@roik.bmf.hu
and
Zoltán Trócsányi
University of Debrecen and Institute of Nuclear Research of the Hungarian Academy of Sciences
H-4001 Debrecen P.O.Box 51, Hungary
Email: Z.Trocsanyi@atomki.hu

***Abstract*—Recent investigations in the field of complex networks include the analysis of distributions of connections in particular networks and the clustering properties of networks. In the first part of my presentation I analyse a network: the dependency graph of the software packages of the Ubuntu distribution of GNU/Linux. It is a directed acyclic graph, and I analyse the degree distribution of the graph for in-degrees (connections directed into a node, the dependent packages), out-degrees (connections directed away from a node) and plain-degrees. I also analyse the clustering properties of the graph. In the second part, I discuss the teaching of these topics at the college level. Using the NetworkX modul for the Python programming language students can easily investigate these properties of the package dependence graph.**

I. Introduction

Recent investigations of complex networks [2] resulted in many interesting observations and discoveries that have potentionally important applications both in social and life sciences. The analytical skills required for such investigations do not involve high-level mathematics, therefore, the new concepts can fairly easily be transferred to the general education.

Computers are popular among young people and are readily available almost everywhere. In order to bring the studies of networks into the classroom, we have choosen a graph, where the nodes are software packages of a freely available operating system (Linux) and the links are dependences among them. The level of the required skills for such investigations are at the college-level, but can be used in special classes in secondary schools as well.

II. Dependences of Linux packages and Python analysis software

There are two well-known package formats for Linux: the *deb* and *rpm* packages, compressed archives of the files of a software compiled for a given architecture with some metadata. The *deb*s have been created for Debian and been used in Linux distributions like Debian [6] and Ubuntu [7]. The *rpm*s have been created by Red Hat Inc. for the Red Hat Linux and are used in some other distributions including Mandriva, SuSE, Fedora as well. In this presentation we concentrate on *deb*s as provided in the Ubuntu 8.04 distribution.

The `apt` package management tool, used mainly for the *deb* packages, is a collection of programs to help install, remove or upgrade packages. The *deb* files are accessed using optical discs or repositories on the Internet and describe the dependencies, so – if we want to install a package – `apt` knows which other packages need to be installed in addition.

We can also use the `apt` program to get the main properties of packages with the `apt-cache` command. For instance, the `apt-cache show vim` command gives the following output:

Package: vim
Priority: optional
Provides: editor
Depends: libc6 (>= 2.7-1), libgpmg1 (>= 1.19.6-1), libncurses5 (>= 5.6+20071006-3), python2.5 (>= 2.5), vim-common (= 1:7.1-138+1ubuntu3), vim-runtime (= 1:7.1-138+1ubuntu3)
Filename: pool/main/v/vim/vim_7.1-138+1ubuntu3_i386.deb
Description: Vi IMproved - enhanced vi editor
Origin: Ubuntu

From this we can see, which version of other packages the `vim` package depends on. All packages have an attribute called priority, which can be required, important, standard, optional and extra; in this case optional. The packages for Ubuntu can be in three repositories: main, universe and multiverse. In our analyis we include files from all. From the row starting with Filename we can conclude, that `vim` is in the main repository and was compiled for Intel 386 architecture.

Python is an object oriented programming language with an extended standars library and some useful modules for science [8]:

- NumPy for matrix calculations,
- Scipy for scientific tasks, for example for optimization and fitting data,
- NetworkX for invesigating network (or graph) properties,
- Matplotlib (and its part pylab), a plotting tool.

We use the IPython, one of the interactiv shells for Python. It has some very convenient features: for example, running from command prompt, automatic identation, input and output history and interactive plotting of diagrams and graphs. We wrote a program called `packages.py` to ma-

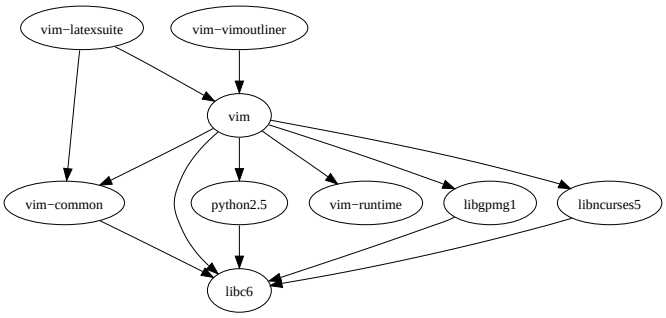


Figure 1. A small part of the dependency graph which includes the vim package and the predecessors and successors of vim.

nipulate and analyse the data of the packages. It is based on the `NetworkX` [9], `apt`, `apt_pkg` and `SciPy` modules. These programs are platform independent except the `apt` and `apt_pkg` modules, that generate the dependency graph from the Ubuntu package database. On other platforms (e.g. Windows or MacOS) we can investigate saved graphs. One can download and analyse saved graphs from the webpage http://mail.roik.bmf.hu/num/complex_networks/packages as `ubuntu_packages*.dot`, where * can be nothing or the date of creation. The `packages.py` file can be downloaded from the same website.

III. Basic properties of the dependency graph

The dependency graph is directed. If package A depends on B, then the (A, B) edge is member of the graph. One package in the graph has successors and predecessors, namely the packages it depends on and the packages that depend on it. In Fig. 1 we can see the vim package with its successors and predecessors. The first row lists the packages needed by the `vim` package: `['vim-vimoutliner', 'vim-latexsuite']`.

The graph is not acyclic because it has some pairs of packages depending on each other. As of July 2008, it has 657 such pairs. Among these 559 pairs are language packages, such as
language-pack-sr-base ↔ language-pack-sr
language-pack-kde-sr ↔ language-pack-kde-sr-base
openoffice.org-hyphenation-sr ↔ language-support-writing-sr

IV. Degree distribution

The simplest model for a complex network is the Erdős–Rényi model defined as N labeled nodes connected by n edges, which are chosen randomly from the set of $N(N-1)/2$ possible edges. Each possible edge is connected with the probability p. In this model the degree distribution has a peak at the value pN.

Real networks – like the internet and citation networks – differ from this model in their degree distribution. Instead of having a peak, these networks show a so called scale-free distribution characterised by a power-law function [1],

$$P(k) \sim k^{-\gamma}. \tag{1}$$

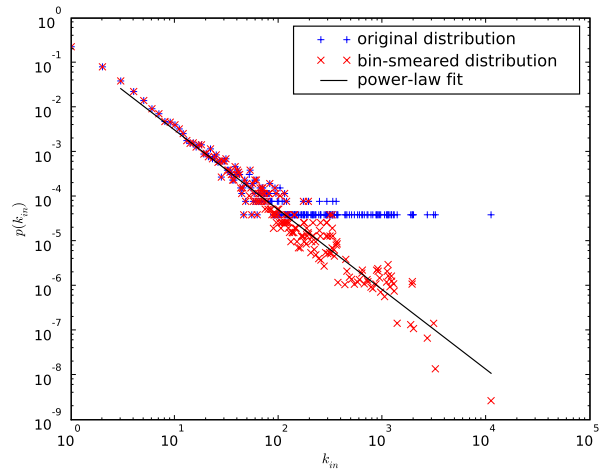
In the case of our dependency graph, we plotted various probability density distributions ($p(k_{\rm in})$ for in-, $p(k_{\rm out})$ for out- and $p(k)$ for plain-degree) in Figure 2. The probability distribution $p(k)$ times the bin width, $p(k)(k_{i+1} - k_i)$ for $k \in [k_i, k_{i+1}]$, gives the probability that the degree of a randomly chosen connection falls in the range $[k_i, k_{i+1}]$ (of course, only integer degrees are possible and not all values are actually realized in a given graph). The blue plus marks show the distributions with unit size bins, $k_{i+1}-k_i = 1$. At a glance, the in-degree distribution exhibits a heavy-tailed power-law shape. In the case of the out-degree distribution there is a kink at $k_{\rm out}\simeq 30$, so it is not clear whether the distribution is best described by a single power-law distribution, or by two distributions with different coefficients. The same feature can also be observed on the plain-degree distribution, although it is suppressed by the effect of in-degrees. To make a decision about the best power-law description, one should make a decision about the fit-range. However, at the tail of the diagram (for large k) there is a lot of degrees with no nodes having that degree. Using logarithmic scales on both axes, the zero values are not visible on the plot. Furthermore, there are many degrees that occur only once or twice. Thus an arbitrary truncation of the fit-range results in a large systematic uncertainty in the fitting of the γ power.

There are known ways to get around this problem [4]. In this analysis we chose to increase the bin size towards the tail of the distribution (bin smearing) such that if the occurences of three degree values k_i, k_j and k_l are non-zero, but there is at least one degree with zero occurence between k_i and k_j, as well as between k_j and k_l and no other degree values with non-zero occurence, then we used the geometric means $k_{j\,\rm min} = \sqrt{k_i k_j}$ and $k_{j\,\rm max} = \sqrt{k_j k_l}$ to define the bin boundaries around the value k_j. Furthermore, we set the probability density value $p(k_j)$ to $p(k_j)/(k_{j\,\rm max} - k_{j\,\rm min})$. These values are plotted with red crosses.

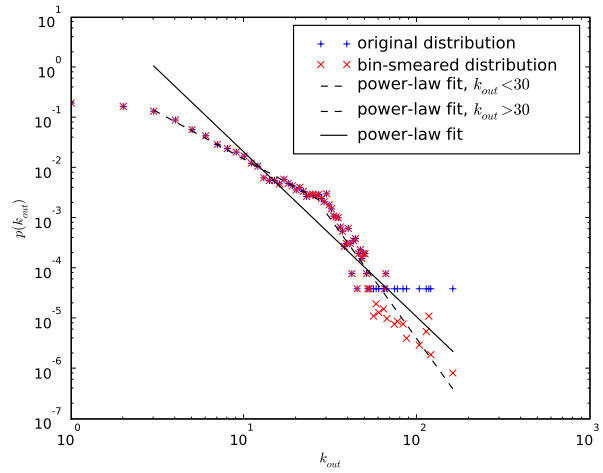
We fitted a power-law function to smeared distributions (dashed line). The exponent is -1.79 for in-degree, -2.88 for out-degree and -2.11 for the plain-degree. For in-degree and plain-degree the fitting is very good, the χ^2/d.o.f. is below 1 % for both cases, while it is much larger, about 7 % for out-degree. The higher value for the latter can be understood from the plot: there is a visible kink in the distribution at $k_{\rm out}\simeq 30$, so the fitted function differs from the data for almost all degrees. We have also carried out fits with different power-law functions for small and large $k_{\rm out}$ values, which resulted in much smaller χ^2/d.o.f. (0.5 % for small and 3.3 % for large values) and very different exponents, -1.87 and -4.79 for small and large values of $k_{\rm out}$, respectively.

Similar investigations to ours were carried out in Ref. [3], where substaintially smaller exponents were found. Although, the network in that study was different from ours, nevertheless, we attribute the difference in the results mainly to the use of unsmeared bins in [3], which yields high sensitivity to the fit range as discussed above.

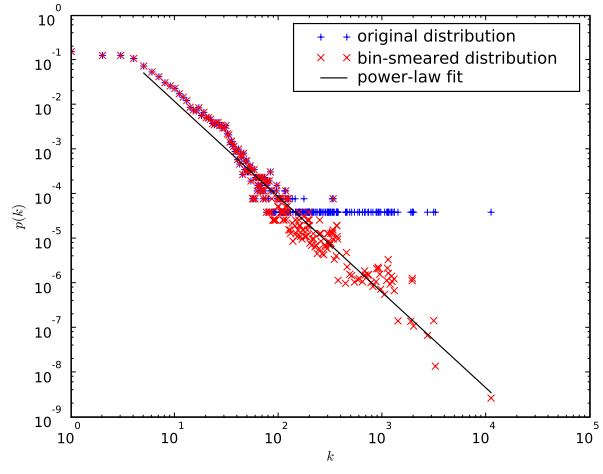
The highest out-degree is 161 (for icthux-desktop), while highest in-degree is 11114 (for libc6 package). We listed other



(a)



(b)



(c)

Figure 2. Probability distributions for (a) in-degree, (b) out-degree, (c) plain-degree (sum of in- and out-degrees). The blue '+' marks correspond to the original, the red '×' to the smeared distribution (see text).

$k_{\rm in}$	package	description
11113	libc6	GNU C Library: Shared libraries
3230	libgcc1	GCC support library
3109	libstdc++6	The GNU Standard C++ Library
2696	libx11-6	X11 client-side library
1985	libglib2.0-0	The GLib library of C routines
1940	zlib1g	Compression library - runtime
1929	perl	Larry Wall's Practical Extraction and Report Language
1865	libxext6	X11 miscellaneous extension library
1381	libgtk2.0-0	The GTK+ graphical user interface library
1296	python	An interactive high-level object-oriented language
1279	libpango1.0-0	Layout and rendering of internationalized text
1226	libcairo2	The Cairo 2D vector graphics library
1217	libatk1.0-0	The ATK accessibility toolkit
1177	dpkg	Package maintenance system for Debian
$k_{\rm out}$	package	description
161	ichthux-desktop	Desktop for Christians
120	ubuntu-desktop	Ubuntu with GNOME desktop environment
117	gobuntu-desktop	
113	ubuntustudio-desktop	
103	texlive-full	Meta package pulling in all components of TeX Live
87	kubuntu-desktop	Ubuntu with KDE desktop environment
83	xubuntu-desktop	Ubuntu with XFCE desktop environment
77	ubuntu-minimal	
74	rhythmbox	Music player for GNOME

Table I
THE PACKAGES WITH LARGEST IN-DEGREES ($k_{\rm in}> 1150$) AND WITH LARGEST OUT-DEGREES ($k_{\rm out}> 70$).

packages with large in- or out-degrees in table I.

V. CLUSTERING COEFFICIENT

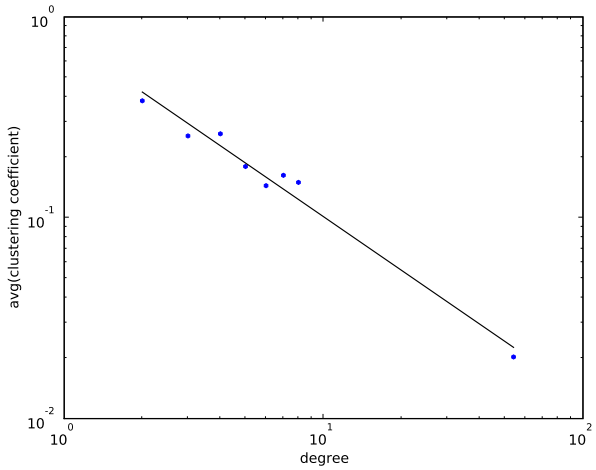
In real networks, like the internet, neighbouring nodes are more likely connected than other two nodes chosen randomly. There is a measure for this property of the graph, called the clustering coefficient. In an undirected simple graph (a graph without multiple and loop edges), if a selected node i has k_i neighbours, then between the k_i neighbours can be no more, than $k_i(k_i-1)/2$ edges. We define the clustering coefficient C_i of a node, as:

$$C_i = \frac{2E_i}{k_i(k_i-1)}, \qquad (2)$$

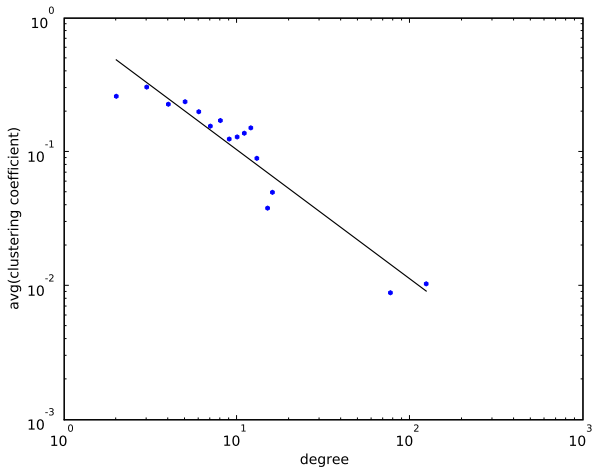
where E_i the actual number of edges between the neighbours [1]. If $C_i = 1$, all pairs of the neighbours are connected. For the whole graph we can define a C clustering coefficient as the average of the C_i-s for all the nodes in the graph. Directed graphs are generally transformed to undirected to calculate clustering coefficients.

In real-world networks C is much greater than in the Erdős–Rényi model, and C_i falls off with k_i, proportional to k^{-1} [5].

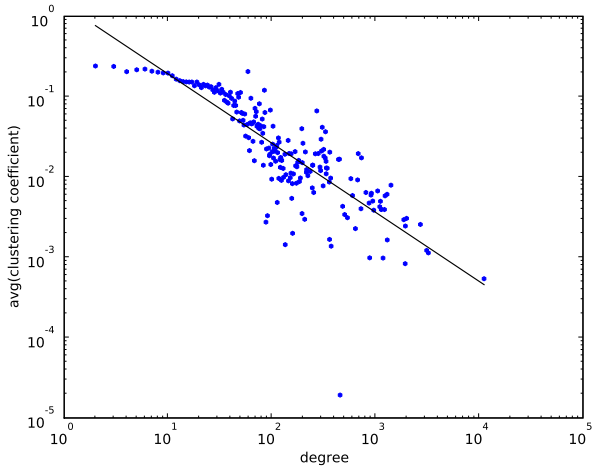
We investigated the clustering coefficient in the whole graph and its two subgraphs. The result is shown in Figure (3). The first subgraph of 69 nodes contained packages with "required"



(a)



(b)



(c)

Figure 3. The average of clustering coefficient as a function of degree. (a) for the packages with required priority ($\gamma = 0.888$), (b) for the packages both from required and important priority ($\gamma = 0.967$), (c) for all the packages we investigated ($\gamma = 0.860$).

Package 1	package 2	N_{pmax}	d_{ij}
libatk1.0-0	libpango1.0-0	1280	0.917
libcairo2	libpango1.0-0	1280	0.904
libatk1.0-0	libcairo2	1226	0.907
libice6	libsm6	1134	0.990
libxrandr2	libxcursor1	906	0.903
libgl1-mesa-glx	libgl1	363	0.922
libxcomposite1	libxdamage1	332	0.938
libqt4-core	libqt4-gui	310	0.916
libglu1-mesa	libglu1	287	0.982
kde-icons-oxygen	kdebase-runtime	174	0.982
kde-icons-oxygen	kdebase-runtime-data	173	1.0
kdebase-runtime	kdebase-runtime-data	174	0.982
libesd0	libesd-alsa0	108	0.990
gnustep-gpbs	gnustep-back0.12	59	0.983
gnustep-gpbs	gnustep-gui-runtime	62	0.935
gnustep-back0.12	libgnustep-gui0.12	65	0.907
gnustep-back0.12	gnustep-gui-runtime	62	0.920
libgnustep-gui0.12	gnustep-gui-runtime	65	0.953
libgnustep-base1.14	gnustep-base-runtime	70	0.971
roxen2	roxen	66	0.924
libblas3gf	libblas.so.3gf	72	0.958
liblapack3gf	liblapack.so.3gf	61	0.950

Table II
THE PACKAGE PAIRS WITH $d_{ij} > 0.9$ AND $N_{pmax} > 50$. (N_{pmax} IS THE MAXIMUM OF THE NUMBER OF PREDECESSORS FOR THE TWO PACKAGES.)

priority (a). Only the libc6 packages had 54 neighbours, the others had less then 9. The second subgraph contained all 161 packages with "required" or "important" priority (b). Here the package libc6 had 124 neighbours, ubuntu-minimal had 77 neighbours and all of the others had less than 17 neighbours. The clustering coefficient as a function of degree in those two cases are close to a power law function with $0.8 < \gamma < 1$. For the last graph, which contained all of the investigated packages (c), for small degrees the exponent is much smaller then for high degrees, sharing this property with some other real networks [5].

VI. DISTANCE BETWEEN PACKAGES

In the previous section, we considered packages being neighbouring if there is a connection between them. We can however, use another measure for defining the nodes to be neighbours. For each package p_i, we collect its predecessors into a set s_i. For each pair of packages p_i and p_j, we mark the the cardinality of the union of the sets s_i and s_j with $|U_{ij}|$ and that of their intersection with $|I_{ij}|$. Then we define a distance measure d_{ij} of the two packages as

$$d_{ij} = \frac{|I_{ij}|}{|U_{ij}|}. \tag{3}$$

If two packages have the same predecessors $d_{ij} = 1$, if the sets of predecessors are disjoint $d_{ij} = 0$. We have searched for packages having many predecessors, with large d_{ij} (Table II).

In the first three rows of the table there is a triple of packages sharing almost all of their predecessors. These are very close together in this sense, however only the packages libpango and libcairo are connected directly (libatk is not connected).

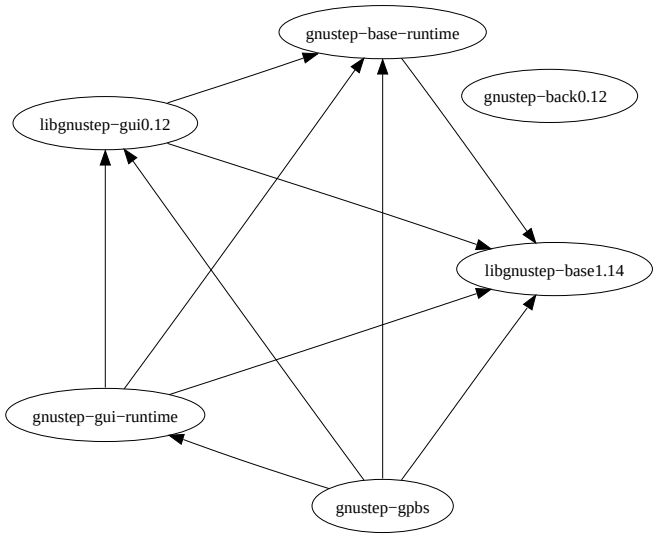


Figure 4. The subgraph of the six gnustep packages.

There is another six rows for packages in connection with gnustep. These share many common predecessors as well. Five of them are connected with edges, but one (gnustep-back0.12) has no connection with them (Figure 4).

In studying the clustering coefficient, we considered nodes that were directly connected. Introducing the distance d_{ij} creates a new possibility to consider nodes to be neighours and calculate the clustering coefficient accordingly. We did not yet pursue this study further.

VII. Bringing the investigation of networks into the classroom

In Section II. we listed the software tools that we used for analysing the properties of the dependency graph. These tools can be used sufficiently easily, so that classrooms excercises can be devised for introducing the properties of complex networks.

The installation of the necessary software is described on the home page (presently in Hungarian) mail.roik.bmf.hu/num/complex_networks/doc/install_hu.html.

In the following we assume Ubuntu platform, the necessary software is installed, the `packages` directory is the current directory, we started the IPython interpreter with the `pylab` option (`ipython -pylab`) and typed the rows below (Python keywords are **highlighted**):

```
import packages
import networkx
```

If the `apt` and `apt_get` modules are installed, we can get the package dependency graph from the cache of the `apt` program. The commands

```
G=packages.get_graph()
print G.info()
```

result in the following list of information:

```
Name:                Ubuntu package dependency...
Type:                MyDiGraph
Number of nodes:     25536
Number of edges:     121316
Average degree:      9.5016
```

On platforms that do not have the `apt` package, we can get a graph from the saved `ubuntu-packages.dot` file skipping the previous step (a text file which uses the DOT language) as well:

```
G=packages.get_graph('ubuntu-packages.dot')
```

We can archive a graph with the `save_graph` method of the object:

```
G.save_graph('2008-07-01.dot')
```

We can select some of the nodes. For instance, the command

```
[node for node in G if 'vim' in node]
```

produces the following list (the original list is longer):

```
['vim-vimoutliner', 'vim-latexsuite',
'vim', 'vim-common', 'vim-full', 'vim-gtk']
```

To identify the predecessors and successors of a certain package, for instance `vim`, we can use the following NetworkX commands:

```
G.predecessors('vim')
G.successors('vim')
```

which was used in creating the graph in Fig. 1.

We can also use the program to draw the diagram of the degree distribution for the package dependency graph. For instance, the code

```
degrees = G.degree()
packages.degree_distribution(degrees,
    direction = 'all',
    savefig_file='deg_dist.png')
```

was used to obtain the graphs in Fig. 2. The direction parameter in the last function may be set to 'in' or 'out' depending whether we want to count only the number of predecessors or successors respectively, 'None' if we wish to analyse the sum of them, or 'all' if we want each these diagrams. If the `savefig_file` is given, the function saves the plot to a file with the given name; eps, jpeg, pdf, png, ps and svg file extensions are allowed. If the direction is 'all', then the directions will be included in the file names, so none of the versions will be overwritten. For example, it saves the file `deg_dist_in.png` for in-degree.

To find the node with the largest in-degree (see end of Sect. IV), we use the commands

```
G.direction = 'in'
G.largest_degrees()
```

VIII. Conclusions and Outlook

In this presentation we discussed some properties of the directed graphs of dependency packages of a freely available operating system, the Ubuntu 8.04 distribution of Linux. We

Model	SW	PDD	CC	Ref.
Erdős–Rényi	no	no	no	[1][4]
Watts–Strogatz	yes	no	no	[1][4]
Albert–Barabási	yes	yes	no	[1][4]
Hierarchical	yes	yes	yes	[5]

Table III
PROPERTIES EXPLAINED BY THE MODELS (SW=SMALL-WORLD PROPERTY, PDD=POWER-LAW DEGREE DISTRIBUTION, CC=CLUSTERING COEFFICIENT AS A FUNCTION OF DEGREE)

have shown that the graphs are scale-free, i.e. the degree distributions are power-law like with exponent -1.79 for in-degree, -2.11 for plain-degree. The out-degree distribution has a kink, exhibiting power-law distributions with exponent -1.87 below $k \simeq 30$ and -4.79 above. If we ignore this kink and fit a single power-law function then we find -2.88 as exponent.

We also found the clustering coefficients C_i of the graph and its certain subgraphs, and found that C_i falls off with k_i proportional to $k^{-\beta}$, with β being close to one, which indicates that these networks are close to being hierarchical.

We also introduced a set of software tools that can easily be employed in the classroom to teach the basic concepts used in studying the properties of complex networks. In addition to the properties of complex networks discussed here, these tools offer further options for investigations in the classroom. In conclusion, we mention three possibilities. These investigations are in progress.

A. The small-world property of complex networks

NetworkX has a function to get the diameter (the maximum length of the shortest path in the graph) of an undirectied graph. Our graph is directed and not connected. Therefore, we need to convert the graph into undirected and also we have to choose the largest connected component with 23760 nodes (as of August 2008). The following piece of code does that:

```
Gu = G.to_undirected()
networkx.is_connected(Gu) # I got False
cc = packages.connected_components()
cc0 = cc0
len(cc0)  # answer: 23760
networkx.diameter()
# answer (after long time): 12

# How many nodes are in
# the 10 biggest connected components?
for i in cc[:10]:
    print len(i),
# answer: 23760 16 12 11 11 8 5 5 5 5
```

For such a big network the running time of this code is quite long.

B. Some generalized random networks and the hierarchical model

We mentioned the Erdős–Rényi model and in the table III we list another three worth to mention. These have detailed description in the papers found in the references.

C. Connections with informatics

Somtimes running programs requires very large CPU time. These days CPU-s with more cores are quite common. In this case, the programmer can split the program into parts. For example, having a dual core CPU, we were able to halve the running time in searching for the common predecessors (Section VI). To get the distances of nodes, the program needed to investigate all (more then 320 million) pairs of packages. With running the program twice in parallel with disjoint set of node pairs, and storing the results, we were able to obtain the whole list of pairs with distances $d_{ij} > 0.2$.

REFERENCES

[1] R. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, p. 47, 2002. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0106096

[2] A.-L. Barabási, "Linked: The new science of networks, perseus, cambridge, ma, 2002," 2002.

[3] N. LaBelle and E. Wallingford, "Inter-package dependency networks in open-source software," 2004. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0411096

[4] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, p. 167, 2003. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0303516

[5] E. Ravasz and A.-L. Barabasi, "Hierarchical organization in complex networks," *Physical Review E*, vol. 67, p. 026112, 2003. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0206130

[6] [Online]. Available: http://www.debian.org/

[7] [Online]. Available: http://www.ubuntu.com/

[8] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[9] [Online]. Available: https://networkx.lanl.gov