Mapping the Open-Source Ecosystem

Ryan Davies (DVSRYA002)

Supervisor: Allan Davids

MPhil. Financial Technology Dissertation

University of Cape Town

DECLARATION

I, Ryan Davies, declare that this thesis is composed of my original work and that it contains no material previously published or written by another person except where due reference has been made to in the text. This thesis is submitted to the University of Cape Town for the Master of Philosophy specialising in Financial Technology degree.

Signed by candidate

Signature

5/7/2023

Date

Table of Contents

1. Acknowledgements

A big thank you to my supervisor Allan Davids for your tireless and enthusiastic support, without your guidance this would never have come close to being completed.

I am also indebted to my parents, Michael and Gillian, who have provided me with the opportunity to undertake this program.

2. Introduction

Unlike private data, which is not traceable, open-source software is traceable and the dependencies between different libraries and modules of open-source software can best be described as a network (Decan, Mens, and Grosjean, 2019). Similarly to academic papers in a field of research, projects in an open-source ecosystem use the work of others and are subsequently used by others as research advances (Bretthauer, 2001). The analog for an author citing the works of another comes in the form of dependencies, defined as when a project imports any code from another project for use within itself. This dissertation uses tools from citation networks and applies them to dependencies in open-source networks taken from the dataset provided by Libraries.io in order to understand the structure and prevailing patterns inherent to open-source ecosystems.

Citation network analysis is a method of research that uses various measures to examine the characteristics of connected pieces of work. It was originally applied in de Solla Price (1965) to bibliographic networks that made use of scientific research papers to introduce concepts like the "research front" which describes the ratio between papers that are cited, cite, or neither in an effort to both highlight critical contributions as well as draw out general trends. Alongside basic metrics such as the number of contributors and citations, an expanded selection of metrics have been developed that include clustering and density to understand the formation and behaviour of these networks (Rose and Georg, 2021). It is noted that different citation networks exist with their own idiosyncrasies and there will be a breakdown at points when comparing a bibliographic citation network to on created by package managers. There is still value in using the likenesses to learn more about the underlying patterns that exist.

The current literature has utilized network analysis within a software context in several ways. There are studies that pick out features within a particular package manager or small group of package managers to gain specific insights (Witten, Suter, and Rajagopalan, 2016). Other studies examine package managers over a period of time to gain insights into how they evolve (Decan, Mens, and

Grosjean, 2019). The application of this knowledge is a priority in the current research, specifically as it pertains to the impact of dependency chains failing (Rahkema and Pfahl, 2022). These pieces of work are part of a growing research field that is supported by the growth in software, and particularly open-source software, over the previous decades (Deshpande and Riehle, 2008). Further research in as new a field as this is required to keep up with changes as the networks expand and mature.

This dissertation primarily aims to broaden the understanding of the trends and analyse a wide range of package managers. Being able to compare across a subject fields and sizes of package managers provides context for the features of each. This is accomplished by using the data from Libraries.io, a web service that tracks various records of 32 package managers. The data includes the names of projects found in each package manager, the projects that depend on them, and the projects that they depend upon. By using this information it is possible to create a network and begin analysis of it. The findings conclude that as the number of projects in a package manager increase, so does the connectedness of the dependency network. This comes with the caveat of an increased amount of centralization in the network. Additionally, there are significant differences in the structure of package managers which can be tied back to their function. This provides a background for both current and future research to be compared against, since much of the current research has been focused on particular package managers or features thereof. The value in knowing how these ecosystems behave in general makes predicting or changing their trajectories more possible.

3. Literature Review

Open-source software has changed the landscape of software development over the last few decades and continues to define a significant amount of how the public engages with software. From operating systems like Linux[1] to development packages like PANDAS[2] to content made available in games through applications like Steam[3], the drive towards making software available for the public has enriched what we are able to access in our own productive aims.

What would become the open-source revolution is chronicled by Bretthauer (2001), documenting both the history involved as well as the guiding principles upon its formation. Starting in the 1970s, multiple operating systems made the leap and became pioneers of making a codebase available to all for no cost. Included in Bretthauer's (2001) analysis is a breakdown of the "cultural and philosophical expectations" that include being able to use the software as desired as well as modify it for one's own purposes. There is an idea of a community that contributes to the upkeep of the project while having a central authority determine which changes are made permanent to the original version. Importantly, the free access and distribution is stressed as a central requirement and strength. One should be able to make changes and then distribute these without any impediment. In combination these traits describe a network of projects that rely upon each other and are freely connected to one another. The question these kinds of networks raise within the context of Bretthauer's (2001) work is just how well the combined efforts live up to the goals set out.

Since the rise of operating systems like Linux and GNU, the trend towards open-source software increased with more and more individuals making their software publicly accessible in an exponential fashion (Deshpande A. & Riehle D, 2008). This increase in quantity and complexity created a requirement for a set of tools that handle this access to a broad range of software. This was solved

[1] Linux is a group of open-source operating systems originally developed in 1991 (https://www.linux.org/).
[2] PANDAS is a package available from the Python package manager PyPI which specializes in data analysis (https://pandas.pydata.org/).
[3] Steam is a distributer of video games and various additions to this software. These additions are often open-source and Steam acts as a package manager for them (https://store.steampowered.com/).

with the invention of the package manager, which provides users with easy access to projects stored on their database often as well as means to navigate the numerous options available. In practice it acts similarly to an “app store” available on smartphones with a simplified user experience through intuitive tools and a centralized store of application. Through this process there was an increase in the ability for developers to use the code of others, publish it, and have that subsequently used by others. This, by definition, increased the compatibility of network analysis since increased connections provide a richer source of data to analyse instead of a sparse collection of unrelated nodes as was previously the case.

Once a network topology is established, the tools to analyse it are required. The field of citation network analysis was first introduced in de Solla Price’s (1965) description of the network created by scientific papers which introduced methods for quantitatively measuring certain characteristics of these networks. While original measures were limited to counts and ratios related to papers that were cited or not cited the field has matured over the years such that many more insightful techniques have been developed. Šubelj, Fiala, and Bajec (2014) investigate three popular bibliographic databases using such techniques. This includes the visual technique of using a “Field bow-tie”, in-degree and out-degree, and clustering profiles to compare databases to one another. Alongside this were counts of citations and other fundamental statistics that are also crucial to establishing a baseline of the network. While the specific content of bibliographic databases is irrelevant to the current topic, the tools used to compare the networks are powerful and provide insight into trends within the networks. Since these measures apply to any network, once it is established that projects in a package manager and their connections are suitable for such analysis it is possible to generalize the measures as well as general approach taken by citation network analysis.

Another investigation on bibliographic networks with useful applications was conducted by Rose and Georg (2021) into the informal networks behind financial economics papers, making use of a broad range of tools from network citation analysis. The paper contains many useful contributions such as

the generation of a novel dataset and domain-specific insights such as a gender imbalance in acknowledgments however the relevant material for this dissertation comes from the measures used to reach these insights. An important element of this work is establishing just how central an author is and this is done by using a variety of "Network Centralities". These include degree, out-degree, and eigenvector centrality which each allow for a look into the value of a single node in the network. Additionally, "to measure and compare different networks in terms of their connectedness", the density and average clustering metrics are introduced (Rose and Georg, 2021). Density measures the number of possible connections made compared to the potential number as a way to gauge how fast information can travel in the network. Average clustering measures how often those who are cited by one source cite each other in order to give a sense of non-linear patterns in the network, for example cases that cite their contemporaries as opposed to forming a single chain of newer and newer additions. Similar analyses exist in the field of healthcare, education and science where a clustering function was used to instead "group together publications with a greater level of association according to the citation networks" providing an insight into the key players in a network (Martinez-Perez, C., Alvarez, C., Villa Collar, C. & Sanchez Tena, M. A., 2020). A related study introduced a metric called the "h-index" to capture the influence of an author in a field (Hirsch 2005) which proves that this analysis is applicable across different fields provided a sufficiently complex citation network. Such derived measures play an important role in nuanced analysis and are combined with summary statistics such at the "Average number of commenters per author" to provide general context as to the nature of the networks (Rose and Georg, 2021). A notable difference between such enquiries into bibliographic networks and the one done in this paper is the complexity of the networks. The scope of information on a software package is limited to how many projects are using it and how many projects it uses in turn while those using papers or articles can leverage more numerous data point such as the number of citations, authors, seminars, conferences, and commenters allowing for a multi-dimensional analysis. Additionally, the networks have been established over a longer period which increases the complexity and general connectedness they have. Consequently, in order to apply a

similar analysis to open-source networks the approach needs to be simplified and adjusted for purpose.

The idea to conceptualize projects in a package manager as nodes in a network is not novel and Decan, Mens, and Grosjean (2019) have proposed several new measures to characterize and compare these networks. They focus on the change in the networks over time and illustrate that some metrics for the networks are more volatile than others. This approach resulted in a broad view of the package managers since each of these would be viewed as a time-series instead of a deep-dive into the data of a particular package manager such as had been done for npm (Wittern, Suter, and Rajagopalan, 2016). The source for the analysis done by Decan, Mens, and Grosjean (2019) was Libraries.io and their work provide a practical example of how to use this valuable service. In terms of their analysis, one of the more novel suggestions was to make use of the Gini index to track the inequality of dependencies or links to certain projects. Normally used on the wealth within a population, in this case they suggest it provides a view into how centralized the network was and also found that the natural shift of the network was to a higher Gini index over time. Another area of focus was looking at the number of transitive dependencies in a system, which were described as the "Achilles' heel" since any dependency transitively depended on by others would be liable to negatively impact a greater share of projects if it were to be removed (Decan, Mens, and Grosjean, 2019). They found that transitive dependencies were common in open-source networks and furthermore went on to provide examples of cases where one of these dependencies breaking was the cause of significant impact on the system. The significance of the risk brought upon by these dependencies is a topic heavily considered in the field as a liability with ways to mitigate it being pursued such as the work done by Rahkema and Pfahl (2022) in the Apple ecosystem where a dataset was compiled for the express purpose of making public vulnerabilities more easily detected. This is related to the investigation of how often packages were updated and how that would affect other projects downstream (Decan, Mens, and Grosjean, 2019). Software packages often undergo changes and this results in a new version being released, however this newer version may use different dependencies

coming in and provide different functionality going out. Combined with the fact that many projects rely on transitive links, this raises a concern for the health of the eco-systems and was investigated. The findings varied across package managers and other dimensions but an important takeaway was that newer projects received more updates. The existence of previous efforts to map the dependency networks (even with different areas of focus) previous as a general blueprint for those wishing to investigate the trends experienced within the open-source software both from a methodological and analytic basis.

Similarly, Kikas et al. (2017) investigated the topological structure of three package managers to understand the risks behind dependencies caused by a popular piece of software propagating bugs throughout a large branch of the network. The three package managers chosen were npm, RubyGems, and Crates which are all large and highly active software ecosystems. A centrally emphasised example of the risk of propagation of faulty software was the package *left-pad* which was removed from npm in 2016 causing a cascade of resulting failures and acted as the alarm bell for such vulnerabilities in projects that depended upon it. In their conclusion they find that over 30% of each of the package managers could be affected by a similar removal of a single important project and recommend the attachment of a “vulnerability score” to alert users to the increasing risk attached. It is concluded that this is a result of the highly central nature of these networks, with a few projects being relied upon by a large proportion of the network. Another conclusion was that there are structural differences between the networks that are highly linked to implicit code changes with changes to the versions of the packages. Finally, the rapid evolution observed of these networks provides reason for certain steps to be taken to improve the user experience by avoiding issues like dependency redundancy. Kikas et al. (2017) use npm as an example of progress in this front due to its ability to download required projects simultaneously but also suggest this analysis of the networks may lead to a more efficient solution. The ability for network analysis to predict and provide suggestions into improving the function of such networks is a strong motivator for more research in the field as such knowledge can lead to actions and policies that improve their health. This paper highlights outlines certain trends

within the three package managers selected that would be useful to see in the context of a larger sampling of package manager.

The current literature investigating the networks present in open-source ecosystems are useful in that it investigates the general trends found in a number of package managers over time. Time is often used as a variable however there is a tendency investigate primarily the largest and most popular package managers which leaves open the opportunity to contrast them next to the smaller options available as well as go into more depth in a static analysis. This paper aims to build upon these advancements in order to further characterize and contrast open-source networks through analysis of the data provided by Libraries.io.

4. Data and Methodology

This dissertation uses statistical tools to investigate the structure of the networks generated by open-source package managers and thus the methodology consisted of ingesting, pre-processing, calculation of, and analysis of the data. Fortunately, the data from Libraries.io is itself open-source which allows for use within any context provided it is correctly cited.

Before any description of the processing undertaken can start, the nomenclature for this paper needs to be defined. The name for each element often varies across package managers with an example being of projects referred to as libraries, packages, or modules. Despite the different wording, the operation and data available is functionally identical which allows for all package managers to be directly compared. With this in mind, the base unit of an open-source network is a *project* which is a named collection of files that have been published to a package manager. *Package managers* are a system of tools that allow for developers to access the projects of others as well as make their own projects accessible. In the context of open-source software, the access to these projects is done without charge and thus allows for novices to have access to advanced libraries without the need to go via an academic institution or pay to use each one. In order for a project to use the code developed by another project, it needs to import the data which is facilitated by the package manager. This allows for the package manager to track which projects are dependent upon each other and creates metadata that is used by services like Libraries.io. For this analysis there are two important pieces of information, the name of the project that is receiving the data and the name of the project that is providing the data. In this context, projects that provide a piece of code are referred to as the *dependents* and the projects receiving the code are referred to as the *dependencies*. A project is dependent on another if it refers to it or has a dependency to that project. Asymmetricities between the distributions of these two define the inequality within the ecosystem and provide insight into how projects are used by developers. For example, whether most dependencies occur in tight clusters for specific purposes or exist in a few universal libraries that are used by almost all projects. Furthermore,

a package manager might have a wide distribution of the number of dependencies each project requires however it may have a narrower distribution of projects that these dependencies rely upon.

It is important to note that multiple package managers can and do exist for the same programming language since they provide an additional layer to the users experience with the language. Users can choose between different package managers however using multiple package managers together is not a generally accepted practice as the purpose of using a package manager is to consolidate the effort of finding projects into a single interface. Thus, the focus in this research is on how individual package managers create citation networks and not on the citation networks that could exist in a combined network of networks if we combined the package managers that exist for a certain programming language.

The dataset accessed through Libraries.io contains 32 packages of which 13 were determined to be of sufficient size to provide insight into general patterns decided through a qualitative analysis of the package managers using the statistics show in Table 1. Smaller package managers showed zero dependencies in some cases and extremely skewed data in others. Further research into these individual ecosystems may prove fruitful in the future, since their idiosyncratic nature may reveal trends in the domain they function within. However, the focus of this dissertation is to draw out more general trends which resulted in selecting the following packages: Atom, CPAN, CRAN, Dub, Elm, Hex, Maven, NPM, NuGet, Packagist, Pub, Pypi, and Rubygems. Each occupies a niche in the open-source world and domain analysis of each provides an explanation of certain features within in (e.g. more or less likely to cluster). Certain package managers are discussed in depth below to dissect more specific trends.

The concept of a network will be briefly outlined here to clarify how an open-source network maps onto the mathematical construct. A network is defined as consisting of two classes of objects called nodes and edges that are connected to each other. The edges may be unidirectional or bidirectional and each node may have as many edges as there are other nodes in the network. In the case of the

networks in question, the projects act as the nodes with the edges being the dependencies. Since a project depends upon another without reciprocity, these edges are unidirectional and since the state of this dependence is to either exist or not exist the weighting is 1.

For the network to be created and analysed, the data is first ingested as a download from Libraries.io after which it is split into multiple files that allowed for efficient reading of the data. There are datasets available that track the same data since 2017 however this dissertation will focus exclusively on the data published on 12 January 2020 and analyse it as a snapshot of the network. The pre-processing for this dataset includes removing dependencies without incomplete information and selecting only relevant information to reduce computational complexity. An interesting nuance is that projects may have different versions that get updated over time (significantly in many cases) and these versions may have different dependencies. Counting all of the dependencies from each version results in the projects with more versions being counted multiple times and thus weighting measures towards whatever characteristic they had. The step taken to correct this was to only use the latest version as this produces a list of dependencies that are most up to date and part of the current snapshot as discussed above.

With cleaned data, the next step was to generate the initial metrics that would guide analysis across the package managers. These included the number of projects in a package manager as well as some more specific metrics that outline how a network functions. One of these was the percentage of nodes connected to the network which was calculated by finding the number of unique names between all projects with dependencies or dependents then dividing this sum by the total number of projects within the entire project manager. This figure then shows which projects have at least one connection of any sort to the network. Another initial measure was the percentage of “Simple Projects” present in the package manager. This was calculated by finding the number of projects with one to three dependencies that also had no dependent projects then dividing that by the total number of projects connected to the network. Dividing by the number of projects connected to the network as opposed

to the number of projects in total is done since this gives an idea of the distribution of the nodes within the network.

Beyond the summary statistics discussed above, this dissertation uses a few more complex measures that provide insight into different behaviours of a network. Since the method of collaboration or dependency creation between each package manager is almost identical the measures used are the same across each which allows for direct comparisons to be made.

These are described below:

The Gini Coefficient

This was originally developed to measure the degree of wealth inequality within a country or demographic (Decan, Mens, and Grosjean, 2019). It ranges from 0 to 1; with 1 indicating perfect inequality (i.e. one person owns all the wealth) and 0 indicating perfect equality (all members have an equal stake). It is determined for the general case of "values" by calculating the ratio between the area under the curve of cumulative values vs. cumulative rank of the values for the actual distribution of values and existing distribution of values (or the Lorenz curve). Given a case of perfect equality these are equal causing the ratio to become 1 and as the distribution tends towards either the higher or lower ranks of values the ratio decreases or increases respectively. There are two uses of it in this context, the first is to see how much a minority of projects are relied upon for a certain package manager and secondly to identify if there are any cases where some specific projects are referred to by a disproportionate number of other projects.

Density

A measure that is simply the ratio of the number edges in a graph to the number of possible edges (Rose and Georg, 2021). This provides an indication of how well developed the connections are between the different projects however there is a temporal element that reduces the expected value of this measure. Since software projects can only depend upon other projects that have already been published a fully connected network is impossible. The value of this measure is to be able to compare between different package managers to get a sense of which ones are more likely to have projects make use of the available resources.

Clustering

This is a similar measure to density that aims to quantify how connected each of the nodes are to one another (Rose and Georg, 2021). In the context of academic papers, it is calculated by counting the number of collaborators on a paper that collaborate with each other outside of that paper. This

number is normalized per network by taking the average of each of the nodes. The use in this dissertation is to track the number of dependencies that depend upon each other. This not only provides an indication of whether a package manager often finds its projects working together in cliques but also can be used on individual projects to identify and reconstruct particular clusters. The practical application of this is to identify sub-fields that all use similar procedures and libraries in their development as opposed to a fairly unconnected eco system where projects are created for individual use cases that are not able to leverage the work of others. This scenario would create a network graph that shows a majority of nodes that are connected to a small number of central projects with a single edge as opposed to a collection of nodes connected in a more cyclical or complex manner. For the purposes of this project, only projects that share a direct connection are considered for the cluster due to computational resource limitations. In other contexts it may be necessary to consider the next layer of connections to get a broader measure of nodes that may be more tangentially connected.

Transitive Dependencies

The number of transitive dependencies in a network refers to the number of projects that are dependent upon a project that is itself dependent upon another project (Kikas et al., 2017). This practical implications of this are discussed above (Rahkema and Pfahl, 2022) and the calculation is done by finding the number of dependent projects a project has that have their own dependents. An example of a project with two transitive dependencies is shown below in Figure 1.

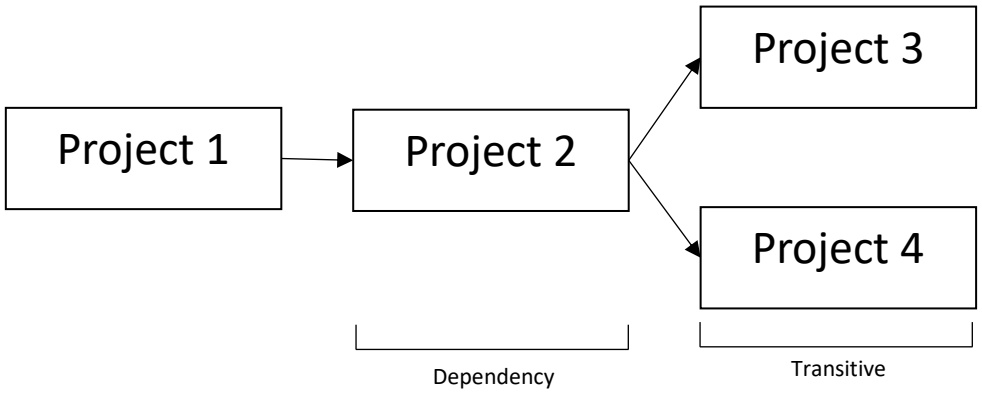

Figure 1: Illustration of Transitive Dependencies

This simple diagram illustrates what a transitive dependency with the example of Project 1 having two transitive dependencies, Projects 3 and 4.

This dissertation only considers the first transitive dependencies so any project that has multiple dependencies chaining off will only have the initial ones counted. This is done to avoid double-counting these dependencies since the calculation will be done on each of the projects. This number is then divided by the number of projects with dependencies to get an average for how many projects with dependents have further linked dependents. This intuitively gives a number that reflects how complex the network becomes after each layer of projects being connected to.

Using this methodology, the following results are generated and the following measures selected to characterize open-source networks for the package managers in question.

5. Results and Analysis

The section covering results and analysis is divided into two parts: the first examines the general trends revealed by the data and the second investigates individual package managers that act as archetypes of specific patterns.

a. Aggregated Results

This part investigates and outlines the general trends found across all the package managers. By looking at a broad range of package managers the contrasts between them can be drawn out as well as the similarities they share.

A few descriptive statistics of general structure of the networks are shown in Table 1, including the total number of projects in each package manager as well as the average and mean number of projects that depend on others. These first steps allow for the most general attributes of the network to be drawn out. A readily observable difference is the varying size of the package managers, ranging from over twelve million for npm to under two thousand for Elm. An important pattern to note is that the average and median number of dependencies is above zero as it provides the knowledge that a typical project in this dataset is connected to the network with at least two edges. If this figure were close to zero it would imply that network analysis is not a useful tool to describe these open-source ecosystems. Another useful statistic (not presented in this table) is that the median of number of dependents for all package managers is 0 which means that most projects are not “cited” by others. Already it is notable that an imbalance between the distributions of projects being depended upon and depending upon has appeared. Finally, a wide range in all the metrics implies that there are meaningful differences between the package managers to be investigated.

Table 1: Summary Statistics of Package Managers

This table shows the total number of projects, average number of dependencies, and median dependencies for all package managers.

Package Manager	Total Projects	Average Dependencies	Median Dependencies
Atom	12847	4.1	2
CPAN	37492	8.7	6
CRAN	16695	7.0	5
Dub	1903	1.9	1
Elm	1505	2.6	2
Hex	9452	2.3	2
Maven	184871	5.5	4
Packagist	313278	3.9	3
Pub	10111	3.5	2
Pypi	231690	3.3	2
Rubygems	161608	4.5	4
NPM	1275082	13.0	7
NuGet	199447	3.67	2

Not every project in the dataset is connected to a network in the same way or even at all. With this in mind, the percentage of projects connected to the network is important since without a significant percentage of connected projects the ecosystem starts to cease being a network entirely and become a collection of unrelated projects contained within the package manager. In this case there is a percentage over 55% for most and significantly above that for many with the notable examples of PyPI and Atom being significantly below this as shown in Table 2 below. The effect and possible reasons for this are discussed in a later section. The last column of this table shows the percentage of projects that have one to three dependencies and no projects dependent on them, termed as “Simple Projects”. The use of this metric is to determine what proportion of each project manager consists of projects that act as basic consumers of information without being developed to a high level of complexity. A higher proportion of these projects indicates a package manager is dominated by the trend of having a radial or star pattern network with these sorts of projects radiating outwards from a highly connected project that acts as a hub.

Table 2: Table showing Percentage of Connected and Simple Projects

This table shows the percentage of connected projects and simple projects for each of the package managers.

Package Manager	% Connected	% Simple projects
Atom	35.9%	72.1%
CPAN	77.3%	22.0%
CRAN	94.4%	28.6%
Dub	56.1%	77.5%
Elm	97.9%	71.8%
Hex	72.5%	85.5%
Maven	59.6%	33.8%
Packagist	58.3%	53.8%
Pub	94.7%	76.1%
Pypi	19.6%	69.8%
Rubygems	85.7%	42.2%
NPM	82.1%	12.8%
NuGet	73.3%	27.2%

There is a link between the number of projects and the average number of dependencies which is quantified by the large correlation of 0.63 between the number of projects and the average number of dependencies which suggests that as the ecosystem grows it makes further connections more likely to materialize, keeping in mind the fact that calculating the correlation in such a small sample set does having its limitations. This correlation is illustrated in Figure 2, plotting the average number of dependencies in a network vs the size of the network as well as the best fit line. This is in concordance with the economic concept of a “Network Effect” which states that the utility of a network increases as the number of nodes in that network increase. This lends support for the idea that open-source projects in the environment of a package manager can best be described as a network. An interpretation of this is that as the package manager grows in size so does the use developers are able to gain from it and this causes more to join and publish projects on it.

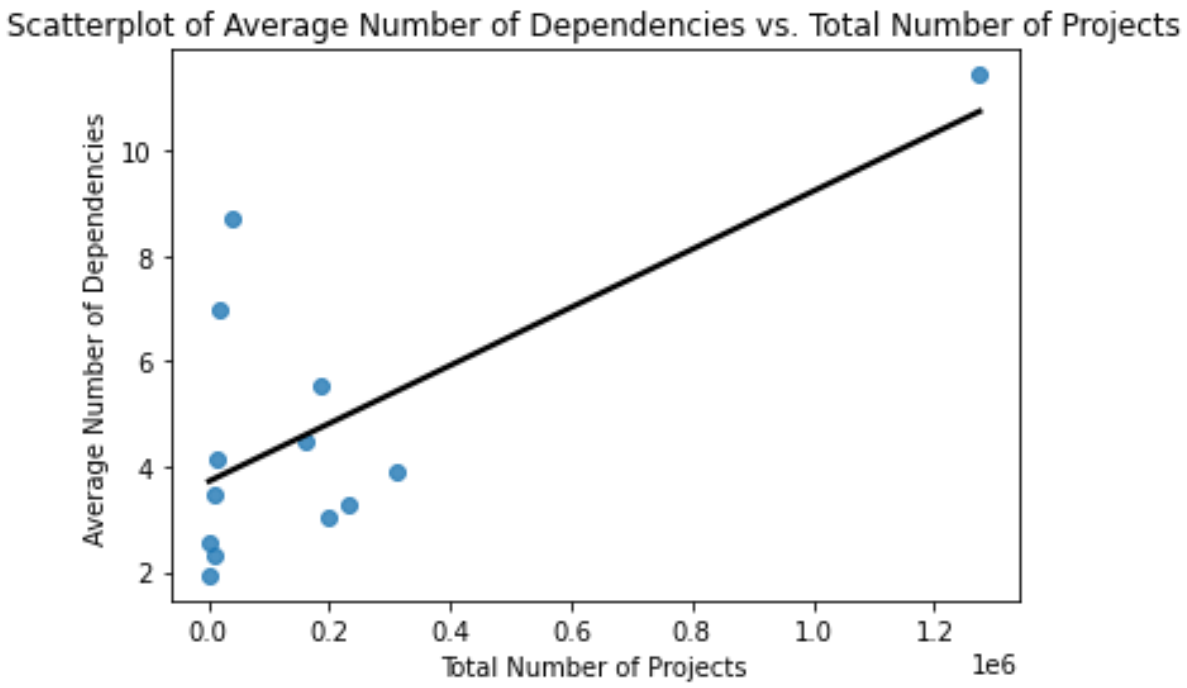

Figure 2: Scatterplot of average number of dependencies vs. total number of projects

This scatterplot is of the average number of dependencies plotted against the total number of projects and has the line of best fit added in black.

A high ratio of dependents to dependencies indicates that more projects are using the same dependent or fewer projects are being used as dependents themselves. Either of these suggest that the network is controlled by a smaller group of projects that provide an outsized influence on the material consumed however the distribution of dependencies amongst these projects is unknown. This Figure 2 shows the broad range between projects, with Dub having an order of magnitude difference between it and npm. The correlation between the size of the package manager and average number of dependencies to total number of projects ratio is -0.30 which shows that as the number of projects increase, there is a weak correlation for the network to become more dominated by the central projects. When combined with the analysis on table 1, it suggests that as package managers become larger the projects tend to be connected other projects more, but this increase is mostly

accounted for by a minority of the projects.

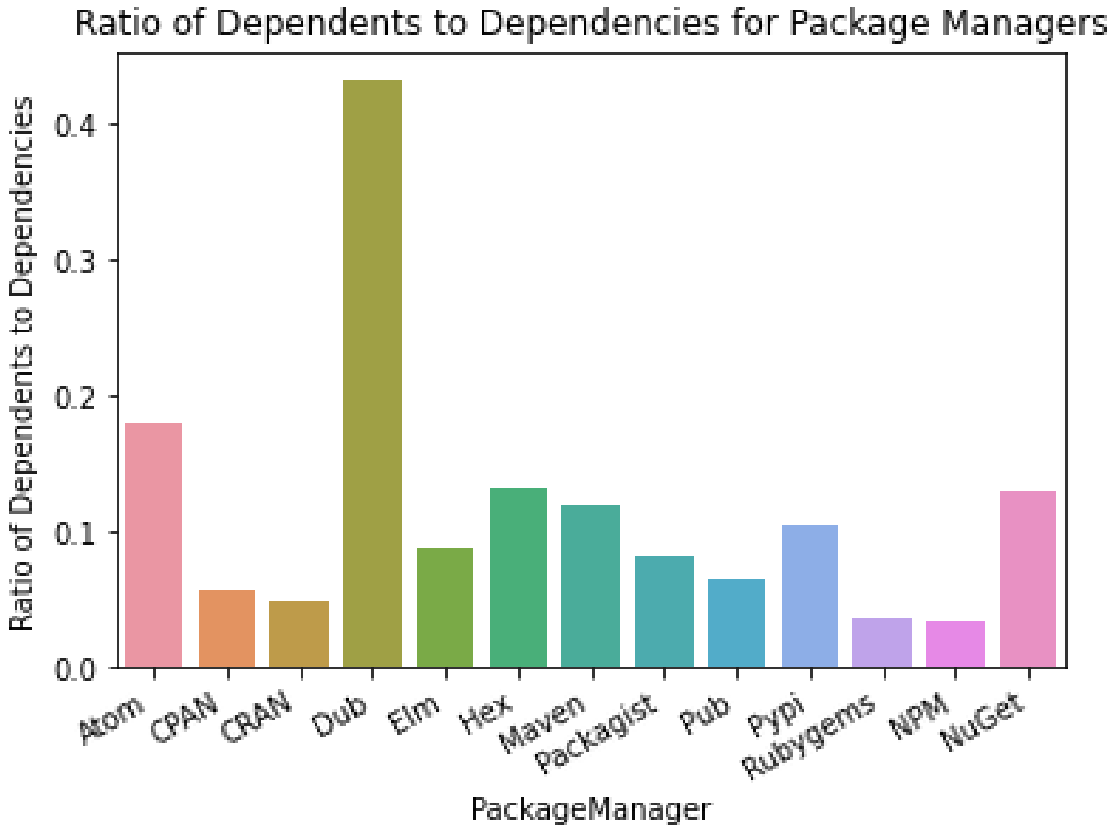

Figure 3: Bar chart of ratio of dependents to dependents for all package managers

This bar chart shows the ratio of dependent projects to the number of dependencies for each package manager.

This tendency to inequality within the network is supported by the Gini Index for each package manager (Table 3). While similar to the measure above, the Gini Index provides further insight into the distribution of these dependent projects. It is possible for a network to be built off a number of dependents that share the dependencies equally while another may have the distribution of number of links are heavily skewed which would result in a more centralized network than the former. The ratio of dependents to dependencies would show both as the same but the Gini Index highlights the importance of the top percentiles. The first observation is that the coefficient is extremely high for all package managers and if it were to be applied in the context of the wealth in a country would be described as severe inequality. This conforms to expectations of a number of crucial packages which dominate while operating in a sub-field, for example within Pypi the packages “Numpy” and “Pandas”

are seen as essential for any data science work which is born out in the fact that over five percent of all projects in PyPI use these two package. As a more general example, the project “Requests” in npm is used in over ten percent of all projects as it allows communication over HTTP. Dub stands out as the package manager with the least inequality however it is important to note that smaller package managers like it can be more subject to effects that inflate or deflate these numbers, for example a large cluster of distinct but closely related projects designed to be used together would have a larger effect on shifting this number than if put amongst the 1.2 million projects in npm.

Table 3: Gini Indexes for Each Package Manager

Package Manager	Gini Index
Atom	0.9966
CPAN	0.9685
CRAN	0.9799
Dub	0.8897
Elm	0.9620
Hex	0.9621
Maven	0.9587
NPM	0.9847
NuGet	0.9556
Packagist	0.9795
Pub	0.9750
PyPI	0.9943
Rubygems	0.9902

Density provides a view into how well connected the network overall is and Figure 4 shows a comparison of this metric between the package managers. Interestingly, it serves to normalize against the size of the package manager since, due to the non-linear increase in possible connections as the network increases, an equal average of connections across two different networks will yield a higher density in the smaller network and this is true even if the number of connections increases in proportion to the different sizes of the network. This process identifies Elm as having a highly connected network and many of the larger package managers such as npm, NuGet, or Packagist are shown to be relatively sparse in their connections. This indicates that while the possible connections and thus theoretical value of the network may increase non-linearly, the number of projects a developer will use does not scale in the same way and that they may have more to choose from but

will not fully realize that value. This is also supported by the fact that in general the smaller package managers serve a smaller number of use cases while a user of the larger one might find themselves in a certain sub-field with little chance of entering any other domain. As an example, CRAN and the R programming language is principally designed for data science and data analysis thus the majority of projects are available as relevant dependents when starting a project while if one were to start a similar project using Python there would be more domains of projects the user would not even consider in PyPI such as the packages used to code gaming features. With this in mind it is no surprise that Elm's function being strictly focused on creating web-based GUIs leads to a higher density.

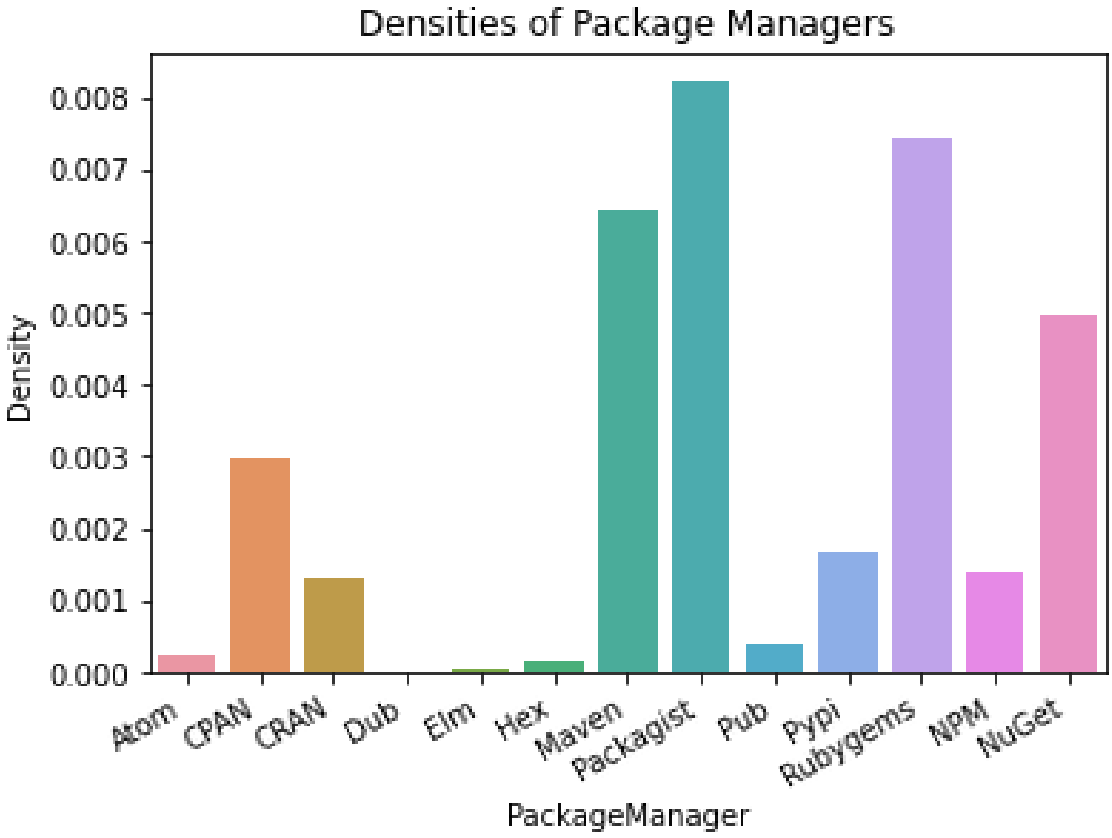

Figure 4: Bar chart of densities for all package managers

This bar chart shows the densities of all the package managers next to each other.

b. Individual Package Managers

This section reviews four individual package managers to investigate trends and archetypes that occur in comparing different package managers using the analysis above. Only a selected number of package managers are able to be given a more rigorous treatment in this section due to the limited scope of this dissertation however by covering four distinct examples it is clear that certain attributes instantiate themselves throughout the samples.

NPM

The first package manager being reviewed is npm (originally known as Node Package Manager) which is used by the Node.js runtime environment. Node.js is a widely used JavaScript framework developed in 2013 that is primarily seen in the web server field and this popularity has led to it having over 1.2 million projects online. Its significance in this context is that it provides an example of a mature and developed network. As postulated earlier, there is a correlation between the size of a package manager and certain trends such as higher inequality and increased dependencies amongst all projects.

The first thing to note is that it has a high number of average dependencies with 13, ranking it first in this category. Notably is also has the fourth highest Gini Index, which shows that while many projects make use of others work this is heavily slanted towards a minority of the projects that dominate the landscape. This is further illustrated by the fact that 0.01% of the projects are the provider of 81.3% of all dependencies with the top 20 projects providing 23.3% of all dependencies. An examination of these projects reveals that they provide general use code that covers testing, linting, and bundling of assets. Combining these two pieces of analysis suggests that the large size of npm has contributed to the increased average number of dependencies as compared to other package managers however most of these dependencies are drawn from a small pool of prolific projects which dominate the field. These projects provide basic tools that are required by a majority of users and npm provides access to these.

Another trend that NPM follows is the increasing number of transitive dependencies as the number of projects increased. These transitive dependencies indicate a more complex network that is characterized by projects adding upon the contributions of others cumulatively to create more advanced projects. There is an average of 16 transitive dependencies per dependent which is the fourth highest value for this figure. This corroborates the analysis above, as the existence of a core of critical projects correlated with high inequality would be used as the source of projects that are initially selected and then further built upon by others.

This trend of increased collaboration in larger network was noted previously and npm stands as the consistent end point of that notion. It suggests that as networks expand and evolve the number as well as intricacy of the connections grows such that a small group of projects become highly used even as projects that have used their code become dependencies for a second layer of dependencies.

PyPI

The second package manager to be individually analysed is Python Package Index (PyPI) which provides Python users with access to packages spanning a range of fields from microcontroller programming to machine learning. It was launched in 2003 and provides access to over 250,000 different projects. It is the project with the highest Gini Coefficient having over 20,000. This highlights it as an example of a network that is controlled by a small number of projects that provide the majority of the network with dependencies.

PyPI has 231,690 projects and the second lowest number of dependent projects per project at 0.04. It's high Gini Coefficient of 99.4 outpaces the trend of larger package managers having increased inequality. This is due to Python being used in many fields as well as the consolidation of smaller packages into larger ones such as PyPI. This distorts the analysis as a user can access what would previously have been multiple packages in through a single one which provides them a broader base of useable functionality from fewer connections. Another example of this is SciPy which originally contained a limited functionality but was progressively built upon such currently contains a number of functions across sub-fields in mathematics and signal processing. This emphasis on certain projects is evidenced by the interesting feature of PyPI having the second lowest percentage of first order transitional dependencies in the top 50 projects well as overall. These are links back to a project that go via another project and are generally an indicator of the maturity of an ecosystem due to the required time and effort for developers to build upon the work of one another. This is despite PyPI being the third largest package manager with a middling number of average dependencies per project. Taking this into account, PyPI exhibits a flatter structure that is significantly linked to nodes in the connected network however these links do not build upon each other as seen in other large package managers like npm and CPAN (8.5 and 9.15 average transitional dependencies per dependency). In other words, PyPI provides more cases of projects requiring a dependency to implement a certain functionality and from there it is less likely that someone would further enhance or grow the capabilities of that package.

Another interesting contributor to its high use of popular projects is the position of Python as an easier programming language to learn which leads to an increased number of novices that go through popular learning channels that tend to focus on a limited number of dependencies. This is evidenced by the fact that PyPI has the highest number of projects with a single dependency which would be associated with simpler projects designed by less experienced developers. Furthermore, it has the lowest proportion of nodes connected to the network with 19.6% which means that the vast majority of projects simply complete a task without the use of other codebases or other codebases using it. This is displayed in Figure 5, showing the sparsely connected projects ranked as the top 100 by number of dependent projects.

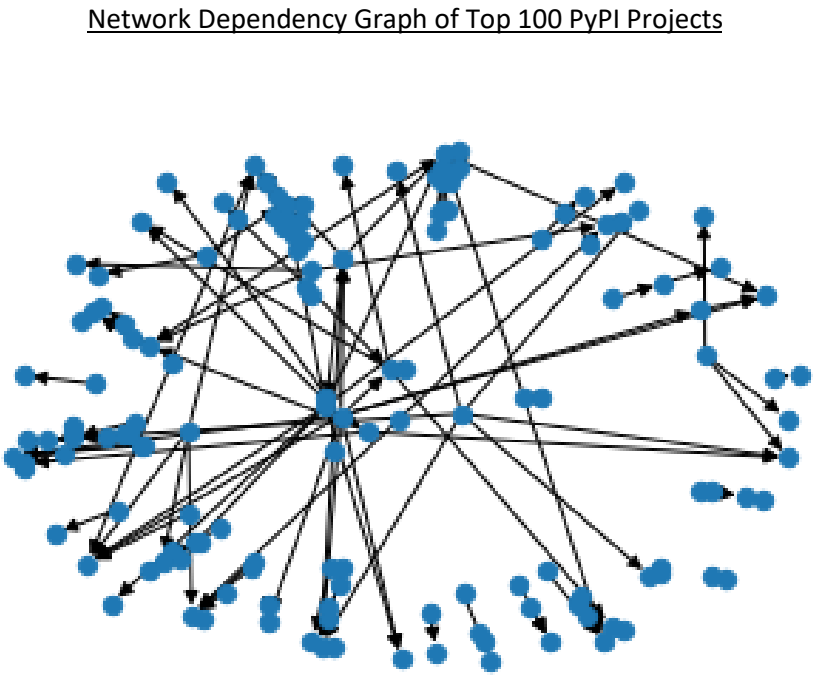

Figure 5: Network dependency graph of the top 100 PyPI projects

This graph shows the top 100 nodes in PyPI as ranked by their number of dependents. These nodes are connected by edges to show any dependencies between each other that they might have.

To conclude, PyPI can be described as a network characterized by both a simple/flat pattern of connections with a number of highly influential projects being linked widely.

CPAN

The Comprehensive Perl Archive Network (CPAN) is a package manager that coordinates projects for the Perl computing language. Perl is a similarly high-level programming language to Python leading to a diversity of applications however its original use of text manipulation remains the most central. Alongside this it finds many uses in networking, such as managing cloud data or the infrastructure associated. It has been active since 1995 and this maturity as well as high active user base has grown it into a complex network with many notable features.

The most apparent are the high average and median dependencies, which rank it second in the package managers reviewed here. The fact that the median sits at just over twice the average of other package managers shows just how well distributed the dependencies are and that the average project is integrated in the network to a high degree. While this suggests a highly connected network of dependencies, the average ratio of number of dependents to dependencies comes out as the fifth lowest which shows that this high distribution on the dependency side is contrasted with a more highly concentration of projects supplying the content to others. Interestingly, despite the relatively smaller base of projects being relied upon, CPAN has the fourth lowest Gini Index which suggests that amongst this group the way the dependencies are distributed is more equal than in other package managers. This follows from the broad uses the language has and that it is not just being used to access a small number of projects for a single use case.

Where CPAN sets itself truly apart is in the high clustering score and the high number of transitional dependencies. Both of these give a measure of the network departing from being a one-way distributer to an interconnected web of projects that leverage each other to a greater extent than other ecosystems. The clustering score of 874 is almost five times the second highest, showing that collaborators share connections in a “horizontal” manner where two projects sharing a same dependency might be connected as opposed to dependencies only coming from the latest out reaches. There were an average of 9 transitive dependencies which puts CPAN significantly ahead of

all but NPM. Transitive dependencies in this case come from the long life of the package manager which gives time for projects to build upon projects that have also built upon projects and so on.

In conclusion, CPAN is the best example of what a network with high levels of complexity and connectedness looks like as the nature of the connections provide both width and depth to a network that builds upon itself both from contemporary and older projects available. This suggests a healthy community behind the network that is aware of what is available and actively contributes.

Atom

Atom is primarily used as a code editor and the package manager function allows for users to import packages that assist in the process of editing such as viewing file icons and improving the aesthetic quality of code. This provides a look into both one of the smaller package managers available and one that has a highly specific use case.

This package manager is the sparsest network encountered of those sampled and shows how a package manager may be more used as a means of distributing packages than a way to facilitate a network of packages that rely upon each other. With 12,827 projects it is the fifth smallest package manager however it has by far the lowest number of projects with dependencies at 102. The second lowest is Elm with 343 which has 1505 projects (88% less) and this results in it having the lowest number of dependent projects per project at 0.007 which is 81% lower than the second place (PyPi). It also has the third lowest density and clustering score, further emphasizing the relative lack of connections within the network. This is considering only the nodes with connections; of the total number of projects only 35.9% of the projects are even connected with a dependency. This means that a vast majority of the projects in this package manager exist by themselves without any connection to the network.

This low number of nodes with dependent projects allows for the visualization of them in their entirety as shown in the graph below:

Network Dependency Graph of Top 100 Atom Projects

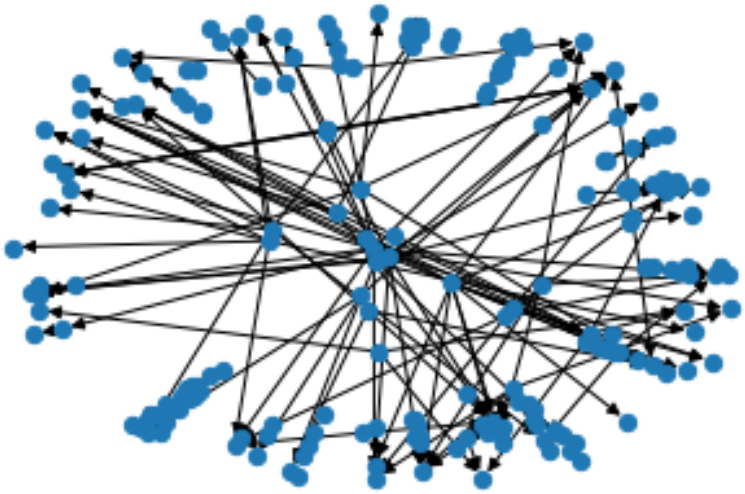

Figure 6: Network dependency graph of Atom

This graph shows the nodes in Atom that have dependent projects. These nodes are connected by edges to show any dependencies between each other that they might have.

Notable about this graph is that there are 102 nodes with only 62 edges, meaning that of the projects with dependencies only 62 receive these from other projects that have dependencies. The rest of the low number of dependencies radiate outwards to projects that are not depended on themselves.

Interestingly, Atom also has the highest Gini index of all the package managers. This can be explained that a small number of package managers with significant popularity would rank highly in any measure of inequality since the other projects have so few connections themselves.

Taken in with the context of how this package manager is used, it can be concluded that Atom finds its niche as a package manager that distributes projects that are relatively simple and do not require the level of interdependencies that more complicated use cases would necessitate. This is congruent with its use of providing simpler plug-ins to make the process of editing code easier as opposed to full scale applications with numerous components requiring dependencies.

6. Conclusion

This dissertation investigated open-source package managers and the networks that they create. By splitting the analysis and results into two sections it was able to show the general trends present across package managers as well as some of the more specific facets shown by individual package managers. In terms of general trends, it was clear that as package managers grow and increase in size the average amount of connections increase however this increase coincides with a pattern of becoming more reliant on fewer projects. This was reported in other papers such as Structure and Evolution of Package Dependency Networks by Kikas et al. (2017) and causes the risk of issues with software propagating far within the network. Individual analysis of four package managers highlighted the differences between them and how an increase in connections could be distributed differently throughout the network. A recurring theme was that how the package manager is used having a predictable impact on the shape of the dependency network such as increasing or decreasing the amount of clustering experienced. Furthermore, each package manager exhibits some difference from the norm and this suggests a need to investigate and map each one individually to extract specific insights that may be useful to those involved with the maintenance and running of the ecosystems. With this in mind, the central claim that package managers could be effectively analysed as a network with projects as the nodes and dependencies as edges has been supported and the resultant quantitative measurements have contributed to the mapping of open-source ecosystems. The implication of this is that the risks associated with them through faulty code or the removal required packages can also be better understood and mitigated through the use of these methods.

The main limitations for this investigation include a lack of computational resources and inability to weight or rank elements in the network. Certain algorithms to generate clustering and transitive link measures would provide a better insight into these dimensions of a network however they currently are prohibitively computationally expensive which excluded them from this analysis. This is due to the fact that these kinds of algorithms seek out an ever widening network of dependents of dependents which becomes exponentially more expensive as the network increases in size. Added to the fact that

the larger networks are the most important to analyse (for example npm, PyPI, NuGet) these measures become less feasible. The use of cloud computing could provide the solution to this issue and make heavy parallelization possible for a reasonable cost. The second limitation is important since it means that every node or edge is seemingly of equal value which is not the case in practice. The Libraries.io dataset does contain a "SourceRank" column however since this would be calculated off the same dataset there would be little information gained. An independent measure of the value of projects would allow for nodes to be differentiated on another measure than just the number of edges they are connected to.

With this step forward completed, it opens the door for further research in this field to understand the patterns occurring more completely. Each package manager could be investigated further due to the vast degree of idiosyncrasies present it each. Elm and Dub would be a highly desirable next steps as they showed an order of magnitude difference in density and the ration of dependents to dependencies respectively. Three areas of analytical processing interest are the impact of highly central nodes on the rest of the network, the identification of possible clusters, and a time sensitive density algorithm. Liu et al. (2011) investigate "driver nodes" that control how a system behaves and it is possible that such an analysis could prevail in this context by using the central core of projects that appear in most networks as the "driver nodes". The impact of this would be to be highlighting which projects are able to have an outsized impact on the system and this would be followed by how they could be influenced to produce the most desirable outcome. Another area involves the algorithmic detection and labelling of clusters in an open-source ecosystem such as that completed by Xiaolong et al. (2008) where they use the Gentoo distribution of Linux to develop models for how the networks evolve. This would probably use unsupervised machine learning techniques to find appropriate clusters which could then be investigated by researchers. An intuitive speculation would be that these clusters may fall into domains of use (e.g. web-based development, data analysis, and visual plotting) however there are many possible axes that such an investigation may take. This dissertation investigated the presence of clustering however there was minimal effort put into

labelling and discovering what impact these clusters had. Finally it would be possible to create a weighted average density that accommodated for the fact that projects cannot create dependencies with other projects that do not exist yet. The current measure uses the total possible number of connections as a comparison to how many connections are implemented however it would be possible to see the number of possible connections at the time a project is added and then average these measures over the lifetime of the package manager. This would provide a more accurate view into what density seeks to measure. The results shown above provide a framework from which to work off and provide more specific and nuanced research of the characteristics of a network in addition to showing which kinds of networks would provide a more or less fruitful avenue for such investigations.

7. References

- Bretthauer, D. 2001. Open Source Software: A History. *Information Technology and Libraries*. 21:3-10.

- Decan, A., Mens, T. & Grosjean, P. 2019. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering.* 24:381–416. DOI:10.1007/s10664-017-9589-y

- Deshpande A. & Riehle D. 2008. The Total Growth of Open Source. *Proceedings of The Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software*. 7-10 September 2008, Milano, Italy. DOI:10.1007/978-0-387-09684-1_16.

- Hirsch JE. 2005. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences of the United States of America.* 15 Nov 2005. 102(46):16569-72. DOI:10.1073/pnas.0507655102

- Kikas, R. Gousios, G. Dumas M. and Pfahl, D. 2017. Structure and Evolution of Package Dependency Networks. *Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. Buenos Aires, Argentina, 2017, pp. 102-112, DOI:10.1109/MSR.2017.55.

- Liu, YY., Slotine, JJ. & Barabási, AL. 2011. Controllability of complex networks. *Nature* 473:167–173. DOI:10.1038/nature10011

- Martinez-Perez, C., Alvarez, C., Villa Collar, C. & Sanchez Tena, M. A. 2020. Current State and Future Trends: A Citation Network Analysis of the Academic Performance Field. *International Journal of Environmental Research and Public Health.* 17(15):5352 DOI:10.3390/ijerph17155352.

- Price, D. J. de S. Networks of Scientific Papers. 1965. *Science*. 149(3683):510-515. DOI:10.1126/science.149.3683.510

- Rahkema K. & Pfahl D. 2022. Dataset: dependency networks of open source libraries available through CocoaPods, carthage and swift PM. *Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22).* Association for Computing Machinery, New York, NY, USA, 393–397. DOI:10.1145/3524842.3528016

- Rose, M. & Georg, C. 2021. What 5,000 acknowledgements tell us about informal collaboration in financial economics. *Research Policy*. Volume 50, Issue 6, DOI:10.1016/j.respol.2021.104236.

- Šubelj, L., Fiala, D. & Bajec, M. 2014. Network-based statistical comparison of citation topology of bibliographic databases. *Sci Rep* 4. Article Number 6496 DOI:10.1038/srep06496

- Wittern E., Suter P., and Rajagopalan S. 2016. A Look at the Dynamics of the JavaScript Package Ecosystem. *Proceedings on the IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. Austin, TX, USA, 2016, pp. 351-361

- Zheng X., Zheng D, Li H., and Wang F. 2008. Analysing open-source software systems as complex networks. *Physica A: Statistical Mechanics and its Applications*. Volume 387, Issue 24. DOI:10.1016/j.physa.2008.06.050

.