The Structural Dimension of Cooperation

Nice subtitle

Jordi Torrents Vivó
jtorrents@milnou.net

November 2016

Director: Joaquim Sempere Carreras

PhD Thesis

Doctorat de Sociologia

Departament de Teoria Sociològica, Filosofia del Dret i Metodologia de les Ciències Socials

Universitat de Barcelona

Contents

List of Figures

List of Tables

Part I

Introduction

1

Introduction

An insightful first sentence. Followed by an insightful first paragraph. and so on and so forth.

1.1 So here we go

Part II

Theory and Methodology

2

The Network Structure of Collaborative Communities

The organization of knowledge production and diffusion has been a challenging problem for economists, sociologists and organization theorists. The increasing importance of knowledge-intensive sectors of the economy, and the inadequacies of markets and hierarchy as coordinating principles for knowledge production and diffusion, has prompted some scholars to suggest that these activities might be better organized through an alternative organizing principle: community (Adler, 2001). It is suggested that a new form of community, qualitatively different from the traditional *Gemeinschaft* and the modern *Gesellschaft* (Tönnies, 1974), has emerged. Examples of collaborative communities are large scientific projects, novel forms of professional work organization (Adler, Kwon, and Heckscher, 2008), open source software communities, and knowledge-intensive production processes in corporations (Adler and Heckscher, 2006).

These collaborative communities are characterized by conscious collaboration, high interdependence, trust, shared values and a value-rational basis for legitimate authority (Adler and Heckscher, 2006; Adler et al., 2008). While all these dimensions matter for a proper characterization of collaborative communities, it is clear that trust plays a more critical role as the key social mechanism of this form. But how does trust develop in these loosely coupled social forms? Adler and Heckscher (2006) suggest that dense local interactions facilitate the emergence of trust, and common values facilitate the development of collective identity. Scarce attention is given to the structural features of collaborations in these communities. Given the sizable literature on the social structure that facilitate (or inhibit) the emergence of trust in society (Granovetter, 1985; Coleman, 1988; Moody and White, 2003), we believe that an important question to further our understanding of collaborative communities is to explore the network structure that lead to their emergence and effectiveness in the production and diffusion of knowledge.

In this chapter therefore we suggest that a unique social network structure undergirds collaborative communities, and facilitate the development of trust and increase their robustness to turnover. Building on the literature on small world and cohesive groups, we identify the key structural feature of these networks which we call cohesive small worlds.

2.1 Collaborative communities

The concept of collaborative communities was introduced to make sense of novel organizational forms which were defying the traditional dichotomy between hierarchy and market (Coase, 1937; Williamson, 1975). Ouchi (1980) was one of the first social theorist to include community/trust in the principles of social organization; he refereed to these principles as clan, but he conceptualized the relation between market, hierarchy and community as a three-way trade-off. Likewise Powell (1990) introduced networks as an alternative principle to hierarchy and markets. Instead, Adler (2001) considers these three principles —Hierarchy, Market and Community— as ideal-types which concrete organizations mix in hybrid forms. Each one of these principles is based on a coordination mechanism. Authority is the main mechanism used in hierarchy to coordinate horizontal and vertical division of labour. Price is the mechanism through which market coordinates competing and anonymous suppliers and buyers. And trust, generated by shared values and norms, is the main mechanism of community principle (Adler, 2001).

This tridimensional space allows a fine gained classification of organizations and institutions, considering the effects of the mixture of different organization principles. Adler and Heckscher (2006) argue that, on the one hand, neither marker nor hierarchy can actually function without at least some underpinning of community and, on the other hand, that the form of community differs depending on its relation to the other two principles of social organization: "When the dominant principle is hierarchy, community takes the form of *Gemeinschaft*. When the dominant principle shifts to market, community mutates from *Gemeinschaft* into *Gesellschaft*. We postulate that when community itself becomes the dominant organizing principle, it will take a form quite different from either *Gemeinschaft* or *Gesellschaft*" (Adler and Heckscher, 2006, 16).

This new form of community can be called collaborative community and it is based on contribution-based trust as its primary social mechanism: "The basis of trust is the degree to which members of the community believe that others have contributions to make towards this shared [end]" (Adler and Heckscher, 2006, 21). This form of community seems especially well suited to deal with the challenges of knowledge-based production processes because, hierarchy and market have proved ineffective, at best, at managing knowledge. On the hierarchy side, knowledge is treated as scarce resource and therefore centralized at the higher levels of the organization where key decisions are taken; this rigid scheme prevents the necessary flexibility to deal with unanticipated problems —very common in non-routine tasks— and to foster innovation and generation of new knowledge (Adler, 2001, 216).

On the market side, the price mechanism fails to optimize the production and allocation of knowledge (Arrow, 1962; Stiglitz, 1996). The fact that knowledge is a public good that grow rather than diminish with use poses serious problems to the effectiveness of price mechanism. There is a trade-off between production and allocation: "On one hand, production of knowledge would be optimized by establishing strong intellectual property rights that create incentives to create knowledge. On the other hand, not only are such rights difficult to enforce, but more fundamentally, they block socially optimal allocation. Allocation of knowledge would be optimized by allowing free access because the marginal cost of supplying another costumer with the same knowledge is close to zero" (Adler, 2001, 217).

In conclusion: "neither markets nor hierarchies [...] nor any intermediate forms [...] can simultaneously optimize incentives to produce knowledge and to disseminate it" (Adler and Heckscher, 2006, 29). But community can effectively deal with knowledge production and distribution by "reduc[ing] both transaction costs –replacing contracts with handshakes— and agency risks —replacing the fear of shirking and misrepresentation with mutual confidence. Community can thus greatly mitigate coordination difficulties created by knowledge's public good character" (Adler and Heckscher, 2006, 30). The community principle of coordination allows to combine different people with different sets of knowledge and expertise in order to solve complex problems while in the process they benefit each other and their common goal.

This solution to the organization of knowledge intensive activity has helped us understand puzzling empirical cases, but few theoretical puzzle need to be addressed. Given the central role that trust play in collaborative communities, it seems essential to explore the conditions in which trust can thrive. Adler and Heckscher (2006) suggest that individuals in collaborative communities will develop higher trust because given the high interdependence of their work, they need to collaborate to achieve their common goal. Furthermore they suggest a common value orientation facilitate the development of common identities. This approach, while based on decades of literature on trust and traditional communities, is problematic for two reasons. First of all, it is not clear how trust and value congruence emerge. Both these characteristics are neither easy to find, nor to maintain, and it is theoretically critical to ask ourselves if there are factors that can explain both. Moreover, it is not clear how trust and shared values can be maintained in large heterogeneous geographically distributed communities where membership exhibits high turnover (a common feature of many concrete collaborative communities).

We argue that the current characterization of collaborative community can be fruitfully enriched with the growing literature on social networks, in order to identify the structural conditions that enable trust and value congruence. A structural approach to collaborative communities is not inconsistent with what has been done so far, but will help (1) refine the current characterization of communities in social network terms, (2) provide a methodology to unobtrusively identify collaborative communities in the wild, and (3) a contribution to the existing tool-kit to design collaborative communities. We explore existing models of network of knowledge production, compare them, and suggest that there is a consistent set of topological properties that characterize collaborative communities.

2.2 A network approach to collaborative communities

A network approach to collaborative communities should start from the basic building block of collaborative activity: team work. Indeed even beyond corporate settings, there is evidence of a trend towards more collaborative activity, often associated with the increasing complexity and interdependence of knowledge production and creative activity more generally (Guimerà, Uzzi, Spiro, and Amaral, 2005; Uzzi and Spiro, 2005; Wuchty, Jones, and Uzzi, 2007; Jones, Wuchty, and Uzzi, 2008). Based on works in science, engineering, social sciences, arts, humanities and patents, Wuchty et al. (2007) show that until the 1950s solo-authored academic articles and inventions were more likely to receive a large number of citations than articles and inventions developed by teams. This is not true anymore, and the trend towards collective

research and teamwork is illustrated by the fact that in the last decade the top cited papers in all the disciplines studied were mostly created by teams.

The study of science as collaborative creative work, and scientific communities as collaborative communities has contributed to our understanding of the properties of the networks created by these collaborations. For instance, Guimerà et al. (2005) suggest that three simple mechanisms of team assembly (number of team members, probability of team members being and incumbent, and propensity to repeat collaborations) determine the topological properties of the network structure that emerge and are correlated with the performance of the teams. They study the evolution of collaboration networks in Social Psychology, Economics, Ecology and Astronomy, and show how the network evolve from a structure characterized by isolated clusters of scientists towards one in which a large portion of them are connected (in networks terms, they all belonged to the same component: all nodes that can be connected to each other by at least one path). In all cases more than half the scientists belonged to the largest connected component of the network. The relative size of the giant component was also associated with performance (publishing in journals with high impact factor) in social psychology and ecology (but not in economics and astronomy). Guimerà et al. (2005) argue that a large connected component in a field would be an evidence of the existence of an invisible college (de Solla Price, 1986; Merton, 1979): a network of social and professional relations linking scientists across universities, which forms a repository of resources and knowledge developed in the past collaborations of the members of the filed. The emergence of a giant component, therefore, seems like a necessary, but clearly not sufficient feature of the network of a collaborative community.

Networks of knowledge production: small world model

One of the key properties of the network structure of a collaborative community should be facilitating an efficient flow of information and ideas among collaborators. The class of network models that most likely fit these requirements is the small-world model (Watts and Strogatz, 1998). Small World networks are characterized by a high level of local density of social ties and short average distances among nodes in the network. More formally,Watts and Strogatz (1998) postulated than 2 measures can be used in order to quantify small world model: average path length (L) and clustering coefficient (CC). L measures the average number of intermediaries between any two nodes of the network, which theoretically means that it is a measurement of how close resources, people and knowledge are in a concrete network. CC is the mean probability that two nodes that are neighbors of the same other node will themselves be neighbors. This measure has been used as proxy for cohesion or closure of networks. The smallworldiness of a network is usually measured with the small world index Q (see appendix A for a formal definition).

Since the publication of Watts and Strogatz's seminal paper, an important stream of empirical studies have analyzed a wide variety of networks, spanning multiple levels of analysis, with the theoretical apparatus of the small world model. For instance, Uzzi and Spiro (2005) analyzed the network of artists who made Broadway musicals from 1945 to 1989. They found a non-linear association between smallworldiness and the financial and artistic performance of the musicals they produced: at low levels of Q the network consists of many unconnected

teams, which inhibits the circulation of new ideas and hinders creativity; as Q increase there are more links among teams and those links are more local cohesive which foster creativity and exchange of ideas. But if Q continues to rise beyond a threshold: "the network increases in connectivity and cohesion to a point at which connectivity homogenizes the pool of creative material while cohesive ties promote common information exchanges, limiting the diversity of the pool of creative material and trapping artists in echo chambers of like minded collaborators" (Uzzi et al., 2007, 87).

Studies conducted on other types of networks have not consistently replicated these findings (for a recent review see Uzzi, Amaral, and Reed-Tsochas, 2007). The inconsistency of the relation between small world structure and performance could be explained by the wide differences in the activities actors were engaged in, by the different measures of performance used, or by the different time frames of the analysis and to differences in the measurement of performance. In addition to these explanations, we would like to stress the fact that the small world model is based on a purely local measure of cohesion (CC). Therefore a high clustering coefficient only means that local teams are highly cohesive, but these teams might not be connected by anything more than a few random connections, and therefore we cannot say anything about the global connectivity of the network. As we will show in the next section, the global connectivity, and the presence of multiple redundant paths among actors, might play a role in explaining the differences in performance between small world structures in different settings. A network can have high global cohesion and connectivity without too much local cohesion, which is what Uzzi argues "traps artists in echo chambers of like minded collaborators".

Another interesting empirical result of studies of scientific collaboration networks, points to the role of specific actors in keeping the network together. Goyal, Van Der Leij, and Moraga-González (2006) show that the global patterns of collaboration among economists from 1970 to 1999 can be modeled as a small world. They also found that a core of interlinked star authors spanned the network shortening otherwise long path lengths. If those brokers were removed from the network, the average path length would rise sharply and the size of the giant connected component will shrink significantly. In the field of sociology Moody (2004) shows that the global patterns of collaborations among authors does not follow a small world model. Furthermore he showed that the cohesion between subfields in sociology does not depend on a core of brokers, and the network did not fragment until all scholars with 10 collaborators were removed from the network.

In addition to facilitating the diffusion of ideas and the combination of diverse skills and pieces of knowledge, teams can also generate common social norms and trust —another essential feature of collaborative communities. To explain how trust can operate beyond the confine of each team we need to explore its structural antecedents.

Trust and social solidarity in networks: the structural cohesion model

Cohesion and social solidarity are central features for collaborative communities, and distinguish them from both ideal-typical hierarchies and markets. These concepts have a long and illustrious history in sociology (Durkheim, 2008) but their precise characterization has been elusive. Much more attention has been focused on its ideational component, which is based

on the members' identification with a collectivity, than on its relational component (Doreian and Fararo, 1998), that is the structure of social relations among members of the group that facilitate the emergence of cohesion. Indeed, even the collaborative community literature focus almost solely on the ideational community, stressing the importance of collective identification.

White and Harary (2001) and Moody and White (2003) developed a robust operationalization of the relational dimension of social solidarity based on the graph-theoretic property of connectivity (Harary, 1969). They propose two equivalent definitions of structural cohesion: "a group's structural cohesion is equal to the minimum number of actors who, if removed from the group, would disconnect the group" and "a group's structural cohesion is equal to the minimum number of independent paths linking each pair of actors in the group" (Moody and White, 2003, 109). These two definitions are equivalent because of Menger's theorem[1].

The starting point of the social cohesion in a group is a state where every actor can reach every other actor through at least one relational path. The formalization of this state in a concrete group is the size of the largest connected component. The emergence of a giant component, therefore, does not just provide the opportunity to access the invisible college (Guimerà et al., 2005), but is also a minimal condition for the development of cohesion. Moody and White (2003) argue that the removal of a few key nodes can affect the flow of knowledge, information and resources in the network. In network terms, a graph is k-connected and is called a k-component if you need to remove at least k nodes to break it into more components. A 2-component, or bicomponent is a component that requires at least 2 nodes to be removed to break down connectivity. Therefore Moody and White (2003) convincingly argue that a biconnected component provides a baseline threshold for strong structural cohesion.

The cohesive structure of a network can be conceptualized as increasingly cohesive groups —called cohesive blocks— nested inside each other. As an example we can think of a group with an highly cohesive core surrounded by a less cohesive periphery (Borgatti and Everett, 2000). A common structural pattern in large networks is an hierarchical nesting of increasingly cohesive groups at low connectivity levels and non-overlapping highly cohesive groups at higher connectivity levels (Moody and White, 2003, 112). Those highly cohesive groups play a key role in the diffusion of the consequences of social interactions among actors in networks (White and Harary, 2001, 355-356). It is usually assumed that the transmission through the network of knowledge, influence and resources generated by social interactions is limited to people 2 or 3 steps away from the initiator of such interactions. In graph theoretic terms, this means that social interactions have a high rate of decay. However, strongly cohesive blocks allow repetition of information and reinforcement of influence because they are characterized by multiple independent pathways that compensate the decay effects of the transmission of knowledge, influence and resources.

This key feature of cohesive groups provides a plausible social mechanism for the emergence and development of trust in collaborative communities. Actors in strongly cohesive groups are able to compare independent perspectives on each other through a variety of paths

[1] A cutset is a set of nodes that, if removed, would break the component into two or more pieces. A graph is k-connected —has node connectivity k— and it is called a k-component if it has no cutset of fewer than k nodes. Menger's theorem states that a k-connected graph also has at least k node-independent paths connecting every pair of nodes.

that flow through distinct sets of intermediaries, which provides multiple independent sources of information about other's characteristics or identity (White and Harary, 2001, 320). Thus, the perception of an individual embedded in such structures of the other members of the group to whom he is not directly linked is filtered by a the perception of a variety of others whom he trusts because is directly linked to them. This mediated perception of the group generates trust at a global scale.

Collaborative community networks: Cohesive Small Worlds

The two models of network topology discussed above provide a solid theoretical starting point in order to analyze the characteristic network structure of collaborative communities, which, we argue, is a key element to understand trust generation and value congruence between highly heterogeneous and interdependent producers in knowledge-based production processes. Table 2.1 summarizes the key dimensions along which we are comparing small world and structural cohesion, and the features we argue are critical for collaborative communities.

	Structural cohesion model	Small world model	Cohesive small world
Cohesion	Global cohesion: focus on cohesive groups formed by nodes linked by node-independent paths	Local Cohesion: Focus on cohesive clusters linked by few edges	Both global and local cohesion
Role of stars	Connectedness and cohesion are not dependent on stars	Connectedness might be highly dependent on stars	Not dependent on stars
Robustness	Resiliency in front of random and targeted removal of nodes	Resiliency on random removal but not necessary on targeted removal of nodes with high degree	Resiliency in front of random and targeted removal of nodes
Trust	Global trust among all nodes of strong cohesive groups	Local trust only among cohesive local neighborhoods	Both local and global trust
Source of trust	Node-independent paths between nodes	Direct links within dense local clusters	Both
Average path length (L)	Implicit: strong cohesive groups must have relative low L	Explicit inclusion of L in the model	Explicit inclusion of L in the model
Diffusion of social interaction	Cohesive groups as amplifiers of signals in networks	No clear mechanism; it is assumed that low L is enough	Cohesive groups as amplifiers

Table 2.1: Comparison of network models for collaborative communities.

We want to highlight that the structural cohesion model and the small world model are not mutually exclusive. A strongly cohesive network could have an average path length com-

parable with its random counterpart while its clustering coefficient is significantly higher. Therefore, there are networks that fit in the intersection between the two models. In order to illustrate this fact, figure 2.1 depicts examples —with toy graphs of 25 nodes— of a network that is structural cohesive but not small world (figure 3.3), a network that is small world but not structurally cohesive (figure 2.1c) and a network that is both structurally cohesive and small world (figure 2.1b). On the lower row of figure 2.1 there are plots of the robustness of each model in front of the deletion of nodes. Those plots depict the size of the giant component divided by the total number of nodes minus the nodes removed in the preceding steps in log scale. Red dots represent targeted removal of nodes, that is, removing nodes starting with nodes of high degree. Blue marks represent random removal of nodes, in each step we chose a node at random and remove it, errorbars represent the standard deviation over 100 runs of random removal of nodes.

The example of a pure structurally cohesive network (figure 3.3) consists in a 2 dimension grid where all nodes of the network form a giant bicomponent but its average path length is significantly higher than a random network with the same number of nodes and edges, and its clustering coefficient is 0 because there are no triangles. Therefore, this network is not a small world. This kind of network is very robust in front of targeted removal of nodes because high degree nodes are in the middle of the grid; after removing all nodes with degree 4 we still have a cycle formed by the outer edges of the original grid.

The example of a pure small world network (figure 2.1c) is inspired in the caveman network proposed by Watts (1999b). It consists in a fully connected core with 20% of the nodes; where each node in this core is connected to a node of a fully connected subgraph of 4 nodes. Thus, the clustering coefficient of this example is significantly higher than its random counterpart but its average path length is almost the same. But, in terms of structural cohesion, the giant bicomponent is formed only by the 20% of the nodes in the core. It should be noticed that despite the fact that this kind of network has more edges than the pure structural cohesive example, its robustness in front of targeted removal of nodes is much lower. As we can see in the robustness plot depicted in figure 2.1f, if we start removing nodes with high degree, the relative size of the giant component shrinks quickly because high degree nodes are in the core of the network, and every node deleted in the core means that the fully connected subgraph of four nodes linked to it will be outside of the giant connected component. Thus, in this example, connectedness is highly dependent on stars (ie high degree nodes).

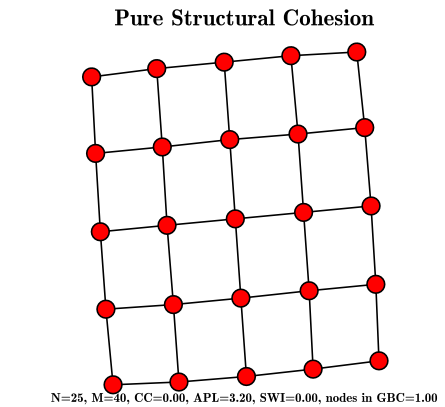
The example of a cohesive small world network (figure 2.1b) is generated algorithmically. We start with a seed formed by a cycle network containing all the nodes in order to make sure that, in the final network, all the nodes will be in a giant bicomponent. Then we randomly link pairs of nodes until we reach the number of edges contained in a 2 dimension grid with the same number of nodes[2]. Then we compute the small world index (Q) of the resultant network —see appendix A for a formal definition— and if it is lower than an arbitrary threshold we start again from the beginning until the resultant network has a small world index greater than

[2]We have chosen to limit the number of edges of the cohesive small world model to the number of edges contained in a 2 dimension grid with the same number of nodes in order to highlight that the density is not the main determinant of the robustness of a network: it is its structure. Thus, a 2 dimension grid is more robust than the cohesive small world model with the same number of edges, and the cohesive small world model is more robust than the pure small world example despite the fact that the latter has more edges

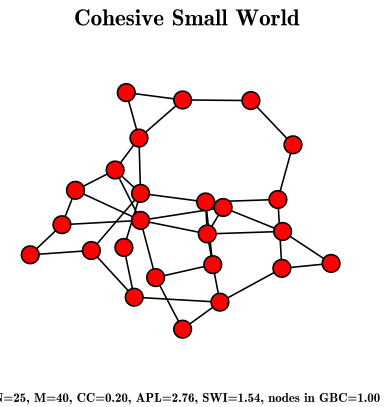
this arbitrary threshold. For the example in figure 2.1b, we have chosen a threshold of 1.5, but any network can be characterized as a small world if $Q > 1$.

Thus, the cohesive small world example has all its nodes in a giant bicomponent —like the pure structural cohesive example— but it also has almost the same average path length than its random counterpart and a clustering coefficient significantly higher —like the pure small world example—. Figure 2.1e depicts its robustness in front of targeted removal of nodes. As we can see, it is in between of the other two examples. We need to remove an important percentage of the nodes with high degree in order to shrink the size of the giant component significantly. Despite the fact that the cohesive small world example has less edges than the pure small world example, its connectedness is much less dependent on stars. Moreover, we need to remove more than 10% of all nodes in order to be able to distinguish the effects of random and targeted removal of nodes in the relative size of the giant component. While in the pure small world example, the effects of targeted and random removal are quite different from the beginning of the removal process.

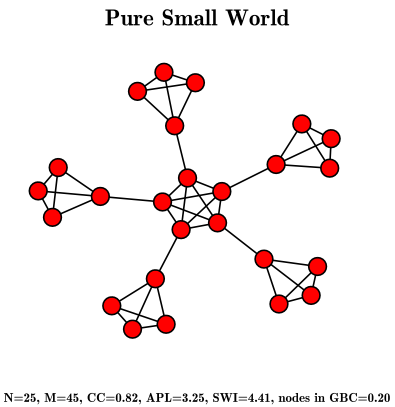
Therefore, we can conclude that the two models are not mutually exclusive. The family of networks that fit in the intersection of both models —what we call cohesive small worlds— exhibit consistent topological patterns. These patterns, we argue, provide the scaffolding for the emergence of collaborative communities. On the one hand, the generation of trust and congruent values among heterogeneous individuals are fostered by structurally cohesive groups in the network that play a key role in amplifying the effects of social interactions trough relatively long paths. On the other hand, the existence of highly connected local clusters allows successful collaboration among heterogeneous individuals with common interests.



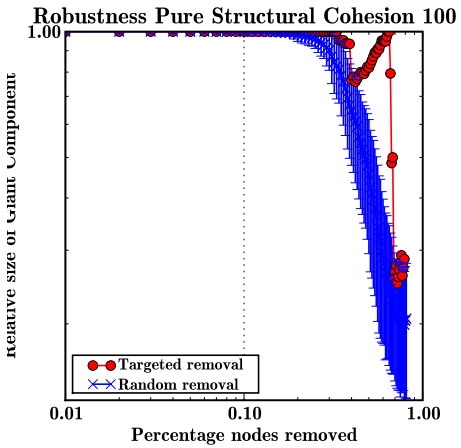
(a) Pure structural cohesion



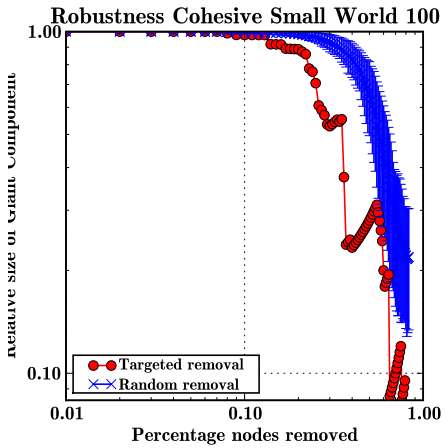
(b) Cohesive small world



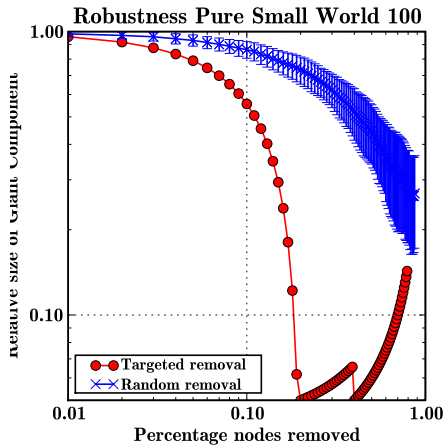
(c) Pure small world



(d) Robustness pure structural cohesion



(e) Robustness cohesive small world



(f) Robustness pure small world

Figure 2.1: Models of network structure and their robustness. The plots of the example networks (figures 3.3, 2.1b and 2.1c) contain 25 nodes in order to facilitate the perception of their structure. The robustness plots (figures 2.1d, 2.1e and 2.1f) are generated by toy networks of 100 nodes because the robustness pattern is clearer than in the 25 nodes examples. We also tested the robustness of all the models with 400 and 2500 nodes and the patterns are the same.

3

Cohesive Groups: The Structural Cohesion Model

Group cohesion is a central concept that has a long and illustrious history in sociology and organization theory, although its precise characterization has remained elusive. Its use in most sociological research has been ambiguous at best. This is largely because, as Moody and White (2003) argued, it is often based on sloppy operationalization grounded mostly in intuition and common sense. Network analysis has provided a large number of solutions to this problem. From classical work in the graph-theoretic sociological tradition on cliques, clans, clubs, k-plexes, k-cores and lambda sets (Wasserman and Faust, 1994, chapter 8), to the more recent contribution of physicists and computer scientists on community analysis (Fortunato, 2010), network theorists have provided researchers with a wide range of measures of cohesion in social networks.

However, neither the classical approaches nor new developments in community analysis are well-enough suited to address many of the common uses of group cohesion in the sociological and organizational literature, for three key reasons. First, while most of these measures can help us identify cohesive subgroups, they do not provide insight into their robustness, which is a critical element to the theoretical conceptualization of cohesion. In most cases, the removal of only a few actors from the subgroups can lead to its fragmentation into smaller disconnected groups (White and Harary, 2001). Secondly, many cohesive subgroup measures do not allow for overlap among subgroups. Finally, even when they do allow for overlap, most measures cannot capture the hierarchical nature of nested social groups, where subgroups, like Russian dolls, are recursively nested in one another. As a result, hardly any of the existing measures capture the theoretical complexity of cohesion, and thus fall short of offering useful operationalizations for many empirical phenomena of sociological interest.

One model which provides a more fertile ground for sociological analysis is the structural cohesion model (White and Harary, 2001; Moody and White, 2003). This model is grounded on two common conceptualizations of group cohesion in the literature. A social group is considered cohesive to the extent that: a) it is resistant to being pulled apart by the removal of some of its members; and b) pairs of its members have multiple direct or indirect connections that pull it together (White and Harary, 2001, 309-310). Building on the concept of node connectivity from graph theory, the structural cohesion of a group is defined in this model as the

minimal number of actors who need to be removed from the group to disconnect it. Despite its solid and elegant mathematical foundation, the structural cohesion model has not been widely used in empirical analysis because it is not possible to perform the required computations for networks with more than a few thousands nodes and edges in a reasonable time frame.

These computational challenges also hindered the development of an interesting feature of the structural cohesion model: its applicability to both bipartite and unipartite networks. While many social networks are essentially bipartite in nature (as people meet, interact, and collaborate around specific events and/or objects), most of our analytical tool-kit was developed to analyze one-mode networks (Latapy, Magnien, and Vecchio, 2008). Therefore it was common practice to conduct network analysis on one-mode projections only, but it is now clear that this practice leads to biased estimates of key measures, as recent work on the clustering coefficient has amply shown (Robins and Alexander, 2004; Lind et al., 2005; Latapy et al., 2008). The structural cohesion model, instead, can be applied without modification to both bipartite and unipartite networks (White, Owen-Smith, Moody, and Powell, 2004). That said, the original algorithm is prohibitively time-consuming to compute, especially with the exponential growth in the size of available network data.

In this chapter we extend the structural cohesion model by using the concept of average node connectivity, that is the *average number of actors who need to be removed from the group to disconnect an arbitrary pair of actors in the group*. We present a set of heuristics to compute structural cohesion based on the fast approximation to compute pairwise node independent paths (White and Newman, 2001). We implemented it in NetworkX (Hagberg et al., 2008), a Python Library for Complex Network Analysis. The heuristics presented here allow us to compute the approximate value of group cohesion for moderately large networks, along with all the hierarchical structures of connectivity levels, one order of magnitude faster than implementations which are currently available. We also suggest a novel graphical representation of the results of the analysis that might help synthetically communicate results and spot differences across different networks (Moody, McFarland, and Bender-deMoll, 2005).

We used our implementation of the heuristics proposed in this chapter to analyze three large collaboration networks: the co-maintenance network of Debian packages, and the co-authorship networks in Nuclear Theory and High-Energy Theory. We ran our analysis in both one-mode and two-mode networks, and compare the networks in terms of their connectivity structure. Consistent with the literature on two-mode networks, we show that the complex hierarchy of collaboration captured in the two-mode analysis is a better representation of the connectivity structure of empirical networks than their one-mode counterparts.

The rest of the chapter is organized as follows: we start by laying out the notation we use in the rest of the paper. Then we discuss the main features which a cohesive subgroup formalization should have from a sociological perspective, reviewing the most important formalizations of cohesive subgroups in the social network literature and discussing in depth the structural cohesion model. We then describe the exact algorithm proposed by Moody and White (2003) to compute the connectivity hierarchy of a given network. After that, we introduce our proposed heuristics, and describe their implementation and performance. We go on to report our findings from applying the structural cohesion analysis to three large collaboration networks, as well as proposing a novel graphical representation of the connectivity structure using a three-dimensional scatter plot. Finally we conclude with implications for future research.

3.1 Terminology and notation

An undirected graph $G = (V, E)$ consists of a set $V(G)$ of n nodes and a set $E(G)$ of m edges, each one linking a pair of nodes. The *order* of G is its number of nodes n and the *size* of G is its number of edges m. Two nodes are adjacent if there is an edge that links them, and this edge is said to be incident with the two nodes it links. A *subgraph* of G is a graph whose nodes and edges are all in G. An *induced subgraph* $G[U]$ is a subgraph defined by a subset of nodes $U \subseteq V(G)$ with all the edges in G that link nodes in U. A subgraph is *maximal* in respect to some property if the addition of more nodes to the subgraph will cause the loss of that property.

A *path* is an alternating sequence of distinct nodes and edges in which each edge is incident with its preceding and following nodes. The length of a path is the number of edges it contains. The *shortest path* between two nodes is a path with the minimum number of edges. The *distance* between any two nodes u and v of G, denoted $d_G(u, v)$, is the length of the shortest path between them. The *diameter* of a graph G, denoted $diam(G)$, is the length of the longest shortest path between any pair of nodes of G. *Node independent paths* are paths between two nodes that share no nodes in common other than their starting and ending nodes. A graph is connected if every pair of nodes is joined at least by one path. A *component* of a graph G is a maximal connected subgraph, which means that there is at least one path between any two nodes in that subgraph.

The *density* of a graph G, denoted $\varrho(G)$, measures how many edges are in set $E(G)$ compared to the maximum possible number of edges among nodes in $V(G)$. Thus, density is calculated as $\varrho(G) = \frac{2m}{n(n-1)}$. A *complete graph* is a graph in which all possible edges are present, so its density is 1. A *clique* is an induced subgraph $G[U]$ formed by a subset of nodes $U \subseteq V(G)$ if, and only if, the induced subgraph $G[U]$ is a complete graph. Thus, there is an edge that links each pair of nodes in a clique. The *degree* of a node v, denoted $deg(v)$, is the number of edges that are incident with v. The minimum degree of a graph G is denoted $\delta(G)$ and it is the smallest degree of a node in G. A k-core of G is a maximal subgraph in which all nodes have degree greater or equal than k; which means that a k-core is a maximal subgraph with the property $\delta \geq k$. The *core number* of a node is the largest value k of a k-core containing that node.

The removal of a node v from G results in a subgraph $G - v$ that does not contain v nor any of its incident edges. The *node connectivity* of a graph G is denoted $\kappa(G)$ and is defined as the minimum number of nodes that must be removed in order to disconnect the graph G. Those nodes that must be removed to disconnect G form a *node cut-set*. If it is only necessary to remove one node to disconnect G, this node is called an *articulation point*. We can also define the *local node connectivity* for two nodes u and v, denoted $\kappa_G(u, v)$, as the minimum number of nodes that must be removed in order to destroy all paths that join u and v in G. Then the *node connectivity* of G is equal to $min\{\kappa_G(u, v) : u, v \in V(G)\}$. Similarly, the *edge connectivity* of a graph G is denoted $\lambda(G)$ and is defined as the minimum number of edges that must be removed in order to disconnect the graph G. The edges that must be removed to disconnect G form an *edge cut-set*.

The measures discussed above are defined as properties of whole graphs but they can also be applied to subgraphs. A *k-component* is a maximal subgraph of a graph G that has, at least,

node connectivity k: we need to remove at least k nodes to break it into more components. The *component number* of a node is the largest value k of a k-component containing that node. Notice that k-components have an inherent hierarchical structure because they are nested in terms of connectivity: a connected graph can contain several 2-components, each of which can contain one or more tricomponents, and so forth.

3.2 Cohesion in social networks

Doreian and Fararo (1998) argue that group cohesion can be divided analytically into an *ideational* component, which is based on the members' identification with a collectivity, and a *relational* component, which is based on connections among members. These connections are, at least in part, observable, and thus the relational approach seems more appropriate for theory building and empirical research. But, despite its attractiveness, the relational component has received much less attention than the ideational component in sociological literature. Social network analysis has been the exception, and since the beginning, its proponents formalized group cohesion in relational terms, that is, they defined the boundaries of subgroups in a community starting from the patterns of relations among actors.

Unfortunately most of the existing formalizations of cohesive subgroups do not capture some key properties of the theoretical concept of cohesive groups. First, a cohesive subgroup should be *robust*, in the sense that its qualification as a group should not be dependent on the actions of a single individual, or any small set of individuals that belong to the group. This implies, on the one hand, that no actor, or small set of actors, should be able to dissolve the cohesive subgroup by abandoning it; while, on the other hand, all actors in a group should be related to all other actors by multiple direct or indirect connections in order to pull it together (White and Harary, 2001; Moody and White, 2003). Therefore, cohesive subgroups should also be relatively invariant to changes outside the group (Brandes and Erlebach, 2005, chapter 6).

Second, actual social groups tend to *overlap* in the sense that some actors are likely to be part of more than one cohesive subgroup. As Freeman (1992) notes, formalizations of subgroups that overlap a lot are not well suited to capturing the theoretical concept of groups because their sociological use is not focused on individuals but on contexts, such as productive relations, friendship relations, or family ties, to name a few. Thus if groups are defined around a higly specific context the overlap is likely to be small. Therefore the formalization of subgroups often assumed non-overlapping subgroups. Moreover, non-overlapping subgroups can be used to develop categorical variables for membership that could be used in regression analysis (Borgatti et al., 1990). However, there is always overlap among cohesive subgroups in actual social groups; and this overlap might be both empirically and theoretically relevant.

Third, following a typical distinction in the social network literature, cohesive groups have both a *structural* and a *positional* dimension. In the former, cohesive subgroups are defined in terms of the global patterns of relations, and the focus is on the groups and the network as a whole. In the latter, the focus is on the identification of actors who, because of their network position, obtain preferential access to information or resources that flow through the

network. Cohesive subgroup formalizations should help address both structural and positional questions.

Last but by no means least, cohesive subgroups are likely to display a *hierarchical structure* in the sense that highly cohesive subgroups are nested inside less cohesive ones. This notion of hierarchy is grounded on Simon's definition: "a system that is composed of interrelated subsystems, each of the latter being, in turn, hierarchic in structure until we reach some lowest level of elementary subsystem" (Simon, 1962, 468). A hierarchical conception of cohesive subgroups implies that there is a relevant organization at all scales of the network, and that cohesive groups are a mesolevel structure that is not reducible to neither macro nor micro level phenomena and dynamics. This nested conception of cohesive subgroups provides a direct link with the structural dimension of the sociological concept of embeddedness (Granovetter, 1985). The nested nature of cohesive groups allows one to operationalize social relations that are, in direct contrast to arms length relations, structurally embedded in a social network.

In the following section we briefly review existing social network formalizations of subgroup cohesion. For each method, in table 3.1 we provide the definition, the underlying logic, the measure proposed, and evaluate them in terms of the four criteria just described. We will therefore consider whether they are robust, can allow for overlapping groups, provide information on both the structure and the position of nodes, and whether they capturethe hierarchical structure of the groups.

Formalizations of cohesive subgroups

Historically, the first social networks approaches to subgroup cohesion formalization identified cohesive subgroups by considering only internal ties among the actors in the group. However, most recent formalizations define cohesive subgroups by considering both internal ties among its members and also external ties between each subgroup and the rest of the network (Wasserman and Faust, 1994). All the formalizations based on internal ties are based on the concept of clique, which were later generalized by relaxing some of the strict conditions of distance, degree or density that the clique concept imposes. The formalizations that consider both internal and external ties can be organized in two main categories depending on whether they use density or connectivity to measure internal and external ties.

The first formalization of cohesive subgroups was the concept of clique (Luce and Perry, 1949), which is a maximal subset of actors in which each actor is directly connected to every other actor in the subgroup. For small groups in some contexts, such as friendship networks, it makes sense to use the clique concept. However, in many contexts, especially in large and/or very sparse networks, it is unlikely that the existing cohesive subgroups will be formed by actors that have direct relations with all other actors in the subgroup. Cliques, however, intuitively capture the idea that a cohesive subgroup exists independently of the action of any individual in the group. Thus the group is robust because it cannot be disconnected by removing any individual actor. Cliques can overlap —and they usually do so a lot— but they do not display a hierarchical organization. Because of the limitations of the clique concept, some generalizations were developed; on the one hand, there emerged a family of generalizations based on relaxing distances among members of the subgroup —n-cliques , n-clans, and

n-clubs (Mokken, 1979); and, on the other, generalizations based on relaxing the number of links between members of the subgroup —k-plex (Seidman and Foster, 1978), and k-cores (Seidman, 1983b).

All these generalizations except for k-core are quite arbitrary because the analyst has to set the parameters n or k depending on the concrete aim of the analysis at hand and its empirical setting. Thus, k-core is the only generalizationof the clique concept with an inherent hierarchical structure: 3-cores are always nested inside 2-cores; and 4-cores inside 3-cores, and so forth. Thus, this formalization captures an important aspect of the sociological concept of cohesive groups. However, k-cores are not robust because the removal of a few actors could potentially disconnect them; in fact they don't even need to be connected at all to be a k-core (White and Harary, 2001). Furthermore, the definition of k-core only considers internal relations among actors within it, without considering relations with the rest of the network.

Another important subset of subgroup formalizations identifies cohesive subgroups by comparing the internal and external ties of subgroups members. The two key criteria to define groups in these categories are density and connectivity. The first formalization of this kind was the LS set (Luccio and Sami, 1969; Lawler, 1973): a set of nodes in which each of its proper subsets has more ties with the nodes outside that subset than the LS set itself. The main idea is that an LS set is a union of subsets of nodes. This union is better than any subset in terms of cohesion because it has fewer connections to the outside. Thus, actors in the LS set have more connections to other members than to outsiders. LS sets are robust to the removal of edges and they have an inherent hierarchical structure; however, due to their strict requirements, only very few LS sets are actually found in empirical social networks. Lambda sets (Borgatti et al., 1990) were introduced as a generalization of LS sets designed to capture only the edge-connectivity properties of the LS sets. Lambda sets are maximal subsets of nodes that have more edge independent paths between them than with nodes outside the subset. This generalization, however, does not capture important features of the sociological concept of group cohesiveness. On the one hand, they are not robust to the removal of nodes, and, on the other hand, the edge independent paths that link the members of a Lambda set can go through nodes that are not in the lambda set, thus there is no strict separation between the role of actors inside and outside a lambda set in respect to its internal cohesion.

	Based on	Criteria	Measure	Definition	Robust	Overlap	Positional	Hierarchical	Computational
Absolute: only internal	complete connectivity	$diam = \varrho = 1$ $\delta = \lambda = \kappa = n-1$	clique	maximal subgraph of nodes all of which are adjacent to each other	Yes	Yes: clique percolation	Yes: structural folds	Yes: k-cliques	Slow
	relax distance	$max\{d_G(u,v)\} \leq n$	n-clique	maximal subgraph in which the largest geodesic distance is no greater than n	No	No	No	No	Slow
		n-clique with $diam \leq n$	n-clan	n-clique that also have a diameter no greater than n	No	Yes	No	No	Slow
		$diam = n$	n-club	a maximal subgraph of diameter n	No	Yes	No	No	Slow
	relax degree	$\delta \geq n - k$	k-plex	maximal subgraph in which each node may be lacking ties to no more than k other nodes	No	Yes	No	No	Slow
		$\delta \geq k$	k-core	maximal subgraph in which all nodes have degree k or more	No	No	No	Yes	Very fast $O(m)$
	relax density	$\varrho \geq \eta$	η-dense subgraph	subgraph with density greater than or equal to η, where $0 \leq \eta \leq 1$	No	No	No	No	Slow
Relative: Internal (+) External (-)	density	minimize edges to outside	LS sets	set of nodes in which each of its proper subsets has more ties with the nodes outside that subset than the LS set itself	Yes	No	Yes	Yes	Slow $O(n^4)$
		quality function of partitions	modularity	the fraction of the edges that fall within the given groups minus the expected such fraction if edges were distributed at random	No	No	No	No	Optimum: Slow Approx: Fast
	connectivity		conductance	weight of edge cut-sets among different subgroups					
		edge-connectivity	lambda sets	maximal subset of nodes that have more edge independent paths between them than with nodes outside the subset	Not as robust as LS sets	No	No	Yes	Slow $O(n^4)$
		node-connectivity	k-components	maximal subgraph that has, at least, node connectivity k: we need to remove at least k nodes to break it into more components	Yes	Yes: $k-1$ nodes	Yes	Yes	Exact: Slow $O(n^4)$ Approx: $\ll O(n^4)$
random walk based partition algorithms					No	No	No	No	Fast

Table 3.1: Summary of cohesive subgroups formalizations from social network analysis literature (Luce and Perry, 1949; Luccio and Sami, 1969; Lawler, 1973; Seidman and Foster, 1978; Mokken, 1979; Seidman, 1983b,a; Borgatti et al., 1990; Wasserman and Faust, 1994; White and Harary, 2001; Moody and White, 2003; Brandes and Erlebach, 2005; Fortunato, 2010). Notation: $diam$ is diameter, ϱ is density, δ is minimum degree, λ is edge-connectivity, κ is node connectivity, n is the number of nodes, m is the number of edges, and $d_G(u,v)$ is the distance between nodes u and v in G.

More recently, under the label community analysis, an interdisciplinary community of researchers interested in complex networks has proposed a novel family of subgroup measures and algorithms (Fortunato, 2010). Essentially their approach is to divide a network into subgroups by grouping nodes that are more densely connected among them than with the rest of the network. To objectively define how good a concrete partition of a network is, they define a quality function (Brandes and Erlebach, 2005; Fortunato, 2010). There are many different quality functions used in network literature, with most of them based on density, but also a few based on connectivity. The most popular quality function is modularity, which is computed as the fraction of the edges that fall within the given groups minus the expected value of the fraction if edges were distributed at random. However, the subgroups resulting from community analysis techniques are not hierarchically organized in the sociological sense discussed above because there is no natural nestedness among groups[1].

The first wave of community analysis focused on the analysis of non overlapping groups, but recent developments have explored overlapping community structures. The most interesting approach of this kind is the clique percolation method (Palla, Derényi, Farkas, and Vicsek, 2005) and their generalizations based on short cycles connectivity (Batagelj and Zaveršnik, 2007). A k-clique is a complete subgraph formed by k members. Two k-cliques are considered adjacent if they share $k-1$ actors. A k-clique community is the largest connected subgraph obtained by the union of all adjacent k-cliques. k-clique communities can share nodes, so overlapping is possible. The clique percolation approach has proven to be a fertile ground over which to build theoretical developments on the positional dimension of cohesion. The concept of intercohesion based on the structural fold network topology (Vedres and Stark, 2010) is the most prominent example. Actors at structural folds are insiders in multiple cohesive subgroups (k-clique communities). Thus they have access to diverse resources and information from each subgroup without being isolated and limited to only one group of neighbors. Vedres and Stark show that this distinctive structural position helps to explain innovation and entrepreneurial dynamics in the context of firm networks.

However these new developments on community analysis are not well suited to address many of the common uses of group cohesion in the sociological literature. The clique percolation method assumes that the network under analysis has a large number of cliques, so it may fail to deliver meaningful results for networks with few cliques; also, if there are too many cliques, it may yield trivial results, such as considering the whole network a cohesive group without internal divisions. Moreover, this method is focused on finding subgraphs that contain many k-cliques inside, which is not exactly the same as subgraphs more densely connected internally than externally, because a k-clique community could be formed by chains of k-cliques with low edge density among non adjacent k-cliques. This implies that k-clique communities are not necessary robust to node removal.

[1] However, some of those methods are called hierarchical because they use hierarchical clustering to organize partitions in each step of the partition algorithm, which is commonly represented by a dendogram. Thus, researchers need to to introduce an arbitrary criteria to identify relevant partitions –that is, the level at which we cut the dendogram.

The structural cohesion model

The structural cohesion approach to subgroup cohesion (White and Harary, 2001; Moody and White, 2003) is grounded on two mathematically equivalent definitions of cohesion that are based on commonly used concepts of cohesion in the sociological literature. On the one hand, the ability of a collectivity to hold together independently of the will of any individual. As set out by the formal definition, "a group's structural cohesion is equal to the minimum number of actors who, if removed from the group, would disconnect the group" (Moody and White, 2003, 109). Yet, on the other hand, a cohesive group has multiple independent relational paths among all pairs of members. According to the formal definition "a group's structural cohesion is equal to the minimum number of independent paths linking each pair of actors in the group" (Moody and White, 2003, 109). These two definitions are mathematically equivalent in terms of the graph theoretic concept of connectivity as defined by Menger's Theorem (White and Harary, 2001, 330), which can be formulated locally: "The minimum node cut set $\kappa(u, v)$ separating a nonadjacent u, v pair of nodes equals the maximum number of node-independent $u - v$ paths"; and globally: "A graph is k-connected if and only if any pair of nodes u, v is joined by at least k node-independent $u - v$ paths". Thus Menger's theorem links with an equivalence relation a structural property of graphs —connectivity based on cut sets— with how graphs are traversed —the number of node independent paths among pairs of different nodes. This equivalence relation has a deep sociological meaning because it allows for the definition of structural cohesion in terms of the difficulty to pull a group apart by removing actors and, at the same time, in terms of multiple relations between actors that keep a group together.

The starting point of cohesion in a social group is a state where every actor can reach every other actor through at least one relational path. The emergence of a giant component —a large set of nodes in a network that have at least one path that links any two nodes— is a minimal condition for the development of group cohesion and social solidarity. Moody and White (2003) argue that, in this situation, the removal of only one node can affect the flow of knowledge, information and resources in a network because there is only one single path that links some parts of the network. Thus, if a network has actors who are articulation points, their role in keeping the network together is critical; and by extension the network can be disconnected by removing them. Moody and White (2003) convincingly argue that biconnectivity provides a baseline threshold for strong structural cohesion in a network because its cohesion does not depend on the presence of any individual actor and the flow of information or resources does not need to pass through a single point to reach any part of the network. Therefore, the concept of robustness is at the core of the structural cohesion approach to subgroup cohesion.

Note that the bicomponent structure of a graph is an exact partition of its edges, which means that each edge belongs to one, and only one, bicomponent; but this is not the case for nodes because k-components can overlap in $k - 1$ nodes. In the case of bicomponents, articulation points belong to all bicomponents that they separate. Thus, this formalization of subgroup cohesion allows limited horizontal overlapping over k-components of the same k. On the other hand, the k-component structure of a network is inherently hierarchical because k-components are nested in terms of connectivity: a connected graph can contain several 2-components, each of which can contain one or more tricomponents, and so forth. This is one

of the bases over which the structural cohesion model is built and it is specially useful for operationalizing the hierarchical conception of nested social groups.

However, one shortcoming of classifying cohesive subgroups only in terms of node connectivity is that k-components of the same k are always considered equally cohesive despite the fact that one of them might be very close to the next connectivity level, while the other might barely qualify as a component of level k (i.e. removing a few edges could reduce the connectivity level to $k-1$). White and Harary (2001) propose to complement node connectivity with the measure of conditional density. If a subgroup has node connectivity k, then its internal density can only vary within a limited range if the subgroup maintains that same level of connectivity. Thus, they propose to combine node connectivity and conditional density to have a continuous measure of cohesion. But connectivity is a better measure than density for measuring cohesion because there is no guarantee that a denser subgroup is more robust to node removal than a sparser one, given that both have the same node connectivity k.

Building on this insight, we propose using another connectivity-based metric to obtain a continuous and more granular measure of cohesion: the average node connectivity. Node connectivity is a measure based on a worst-case scenario in the sense that to actually break apart a k connected graph by only removing k nodes we have to carefully choose which nodes to remove. Recent work on network robustness and reliability (Albert, Jeong, and Barabási, 2000; Dodds, Watts, and Sabel, 2003) use as the main benchmark for robustness the tolerance to the random or targeted removal of nodes by degree; it is unlikely that by using either of these attack tactics we could disconnect a k connected graph by only removing k nodes. Thus node connectivity does not reflect the typical impact of removing nodes in the global connectivity of a graph G. Beineke, Oellermann, and Pippert (2002) propose the measure of *average node connectivity* of G, denoted $\bar{\kappa}(G)$, defined as the sum of local node connectivity between all pairs of different nodes of G divided by the number of distinct pairs of nodes. Or put more formally:

$$\bar{\kappa}(G) = \frac{\sum_{u,v} \kappa_G(u,v)}{\binom{n}{2}} \tag{3.1}$$

Where n is the number of nodes of G. In contrast to node connectivity κ, which is the minimum number of nodes whose removal disconnects some pairs of nodes, the average connectivity $\bar{\kappa}(G)$ is the expected minimal number of nodes that must be removed in order to disconnect an arbitrary pair of nodes of G. For any graph G it holds that $\bar{\kappa}(G) \geq \kappa(G)$. As Beineke et al. show, average connectivity does not increase only with the increase in the number of edges: graphs with the same number of nodes and edges, and the same degree for each node can have different average connectivity (Beineke et al., 2002, figure 2, 33). Thus, this continuous measure of cohesion doesn't have the shortcomings of conditional density to measure the robustness of the cohesive subgroups.

The relation between node connectivity and average node connectivity is analog to the relation between diameter and average distance. The diameter of a graph G is the maximum distance between any two nodes of G, and like node connectivity, it is a worst-case scenario. It does not reflect the typical distance that separates most pairs of nodes in G. When modeling distances between actors in networks, it is better to use the average path length (L) because it

is close to the typical case: if we choose at random two nodes from a network, it is more likely that their distance is closer to the average than to the maximum distance. Taking into account the average connectivity of each one of the k-components of a network allows a more fine grained conception of structural cohesion because, in addition to considering the minimum number of nodes that must be removed in order to disconnect a subgroup, we also consider the number of nodes that, on average, have to be removed to actually disconnect an arbitrary pair of nodes of the subgroup. The latter is a better measure of subgroup robustness than the departure of key individuals from the network.

Structural cohesion is a powerful explanatory factor for a wide variety of interesting empirical social phenomena. It can be used to explain, for instance: the likelihood of building alliances and partnerships among biotech firms (Powell et al., 2005); how positions in the connectivity structure of the Indian inter-organizational ownership network are associated with demographic features (age and industry); and differences in the extent to which firms engage in multiplex and high-value exchanges (Mani and Moody, 2014). Social cohesion can also help us understand degrees of school attachment and academic performance in young people, as well as the tendency of firms to enroll in similar political activity behaviors (Moody and White, 2003). It offers insight, also, into emerging trust relations among neighborhood residents or the hiring relations among top level US graduate programs (Grannis, 2009). In addition to social solidarity and group cohesion, the model can equally fit many relevant theoretical issues, such as conceptualizing structural differences among fields and organizations (White et al., 2004), operationalizing the structural component of social embeddedness (Granovetter, 1985; Moody, 2004), explaining the role of highly connected subgroups in boosting diffusion in social networks without a high rate of decay (Moody, 2004; White and Harary, 2001), or highlighting the complexity and diversity of the structure of real world markets beyond stylized one-dimensional characterizations of the market (Mani and Moody, 2014).

Despite all its merits, the structural cohesion model has not been widely applied to empirical analysis because it is not practical to compute it for networks with more than a few thousands nodes and edges due to its computational complexity. What's more, it is not implemented in most popular network analysis software packages. In the next section, we will review the existing algorithm to compute the k-component structure for a given network, before introducing our heuristics to speed up the computation.

3.3 Existing algorithms for computing k-component structure

Moody and White (2003, appendix A) provide an algorithm for identifying k-components in a network, which is based on the Kanevsky (1993) algorithm for finding all minimum-size node cut-sets of a graph; i.e. the set (or sets) of nodes of cardinality k that, if removed, would break the network into more connected components. The algorithm consists of 4 steps:

1. Identify the node connectivity, k, of the input graph using flow-based connectivity algorithms (Brandes and Erlebach, 2005, chapter 7).

2. Identify all k-cutsets at the current level of connectivity using the Kanevsky (1993) algorithm.

3. Generate new graph components based on the removal of these cutsets (nodes in the cutset belong to both sides of the induced cut).

4. If the graph is neither complete nor trivial, return to 1; otherwise end.

As the authors note, one of the main strengths of the structural cohesion approach is that it is theoretically applicable to both small and large groups, which contrasts with the historical focus of the literature on small groups when dealing with cohesion. But the fact that this concept and the algorithm proposed by the authors, are theoretically applicable to large groups does not mean that this would be a practical approach for analyzing the structural cohesion on large social networks [2].

The equivalence relation established by Menger's theorem between node cut sets and node independent paths can be useful to compute connectivity in practical cases but both measures are almost equally hard to compute if we want an exact solution. However, White and Newman (2001) proposed a fast approximation algorithm for finding good lower bounds of the number of node independent paths between two nodes. This smart algorithm is based on the idea of searching paths between two nodes, marking the nodes of the path as "used" and searching for more paths that do not include nodes already marked. But instead of trying all possible paths without order, this algorithm considers only the shortest paths: it finds node independent paths between two nodes by computing their shortest path, marking the nodes of the path found as "used" and then searching other shortest paths excluding the nodes marked as "used" until no more paths exist. Because finding the shortest paths is faster than finding other kinds of paths, this algorithm runs quite fast, but is not exact because a shortest path could use nodes that, if the path were longer, may belong to two different node independent paths (White and Newman, 2001, section III). Therefore a condition for the use of this approximation algorithm would be that the networks analyzed should be sparse; this will reduce its inaccuracy because it will be less likely that a shorter path uses nodes that could belong to two or more longer node independent paths.

White and Newman suggest that this algorithm could be used to find k-components. First one should compute the node independent paths between all pairs of different nodes of the graph. Then build an auxiliary graph in which two nodes are linked if they have at least k node independent paths connecting them. The induced subgraph of all nodes of each connected component of the auxiliary graph form an extra-cohesive block of level k (like a k-component but with the difference that not all node independent paths run entirely inside the subgraph). Finally, we could approximate the k-component structure of a graph by successive iterations of this procedure.

However, there are a few problems with this approach. First, a k-component is defined as a maximal subgraph in which all pairs of different nodes have, at least, k node independent paths between them. If we rely on the connected components of the auxiliary graph as

[2] The fastest implementation of this algorithm runs in $O(N^4)$ time (Csárdi and Nepusz, 2006) which is impractical for moderately large networks.

proposed by White and Newman (2001) we will include in a given k-component all nodes that have at least k node independent paths with *only one* other node of the subgraph. Thus, the cohesive subgraphs detected won't have to be k-components as defined in graph theory. Second, k-components can overlap in $k-1$ nodes. If we only consider connected components (i.e. 1-components) in the auxiliary graph, we will not be able to distinguish overlapping k-components. Finally, the approach proposed by White and Newman is not practical in computational terms for large networks because of its recursive nature and because it needs to compute node independent paths for all pairs of different nodes in the network as starting point.

3.4 Heuristics for computing k-components and their average connectivity

The logic of the algorithm presented here is based on repeatedly applying fast algorithms for k-cores (Batagelj and Zaveršnik, 2011) and biconnected components (Tarjan, 1972) in order to narrow down the number of pairs of different nodes over which we have to compute their local node connectivity for building the auxiliary graph in which two nodes are linked if they have at least k node independent paths connecting them. We follow the classical insight that, "k-cores can be regarded as seedbeds, within which we can expect highly cohesive subsets to be found" Seidman (1983b, 281). More formally, our approach is based on Whitney's theorem (White and Harary, 2001, 328), which states an inclusion relation among node connectivity $\kappa(G)$, edge connectivity $\lambda(G)$ and minimum degree $\delta(G)$ for any graph G:

$$\kappa(G) \leq \lambda(G) \leq \delta(G) \tag{3.2}$$

This theorem implies that every k-component is nested inside a k-edge-component, which in turn, is contained in a k-core. This approach, unlike the proposal of White and Newman (2001), does not require computing node independent paths for all pairs of different nodes as a starting point, thus saving an important amount of computation. Moreover it does not require recursively applying the same procedure over each subgraph. In our approach we only have to compute node independent paths among pairs of different nodes in each biconnected part of each k-core, and repeat this procedure for each k from 3 to the maximal core number of a node in the input network.

The aim of the heuristics presented here is to provide a fast and reasonably accurate way of analyzing the cohesive structure of empirical networks of thousands of nodes and edges. As we have seen, k-components are the cornerstone of structural cohesion analysis. But they are very expensive to compute. Our approach consists of computing extra-cohesive blocks of level k for each biconnected component of a k-core. Extra-cohesive blocks are a relaxation of the k-component concept in which not all node independent paths among pairs of different nodes have to run entirely inside the subgraph. Thus, there is no guarantee that an extra-cohesive block of level k actually has node connectivity k. We introduce an additional constraint to the extra-cohesive block concept in order to approximate k-components: our algorithm computes extra-cohesive blocks of level k that are also k-cores by themselves in

G. Based on several tests with synthetic and empirical networks presented below, we show that usually extra-cohesive blocks detected by our algorithm have indeed node connectivity k. Futhermore, extra-cohesive blocks maintain high requirements in terms of multiconnectivity and robustness, thus conserving the most interesting properties from a sociological perspective on the structure of social groups.

Combining this logic with three observations about the auxiliary graph H allows us to design a new algorithm for finding extra-cohesive blocks in each biconnected component of a k-core, that can either be exact but slow —using flow-based algorithms for local node connectivity (Brandes and Erlebach, 2005, Chapter 7)— or fast and approximate, giving a lower bound with certificate of the composition and the connectivity of extra-cohesive blocks — using White and Newman (2001) approximation for local node connectivity. Once we have a fast way to compute extra-cohesive blocks, we can approximate k-components by imposing that the induced subgraph of the nodes that form an extra-cohesive block of G have to also be a k-core in G.

Let H be the auxiliary graph in which two nodes are linked if they have at least k node independent paths connecting them in each of the biconnected components of the core of level k of original graph G (for $k > 2$). The first observation is that complete subgraphs in H (H_{clique}) have a one to one correspondence with subgraphs of G in which each node is connected to every other node in the subgraph for at least k node independent paths. Thus, we have to search for cliques in H in order to discover extra-cohesive blocks in G.

The second observation is that an H_{clique} of order n is also a core of level $n-1$ (all nodes have core number $n-1$), and the degree of all nodes is also $n-1$. The auxiliary graph H is usually very dense, because we build a different H for each biconnected part of the core subgraph of level k of the input graph G. In this kind of network big clusters of almost fully connected nodes are very common. Thus, in order to search for cliques in H we can do the following:

1. For each core number value c_{value} in each biconnected component of H:
2. Build a subgraph $H_{candidate}$ of H induced by the nodes that have *exactly* core number c_{value}. Note that this is different than building a k-core, which is a subgraph induced by all nodes with core number *greater or equal than* c_{value}.
3. If $H_{candidate}$ has order $c_{value}+1$ then it is a clique and all nodes will have degree $n-1$. Return the clique and continue with the following candidate.
4. If this is not the case, then some nodes will have degree $< n-1$. Remove all nodes with minimum degree from $H_{candidate}$.
5. If the graph is trivial or empty, continue with the following candidate. Or otherwise recompute the core number for each node and go to 3.

Finally, the third observation is that if two k-components of different order overlap, the nodes that overlap belong to both cliques in H and will have core numbers equal to all other nodes in the bigger clique. Thus, we can account for possible overlap when building subgraphs

$H_{candidate}$ (induced by the nodes that have *exactly* core number c_{value}) by also adding to the candidate subgraph the nodes in H that are connected to all nodes that have *exactly* core number c_{value}. Also, if we sort the subgraphs $H_{candidate}$ in reverse order (starting from the biggest), we can skip checking for possible overlap for the biggest.

Based on these three observations, our heuristics for approximating the cohesive structure of a network and the average connectivity of each individual block, consists of:

Let G be the input graph. Compute the core number of each node in G. For each k from 3 to the maximum core number build a k-core subgraph G_{k-core} with all nodes in G with core level $\geq k$.

For each biconnected component of G_{k-core}:

1. Compute local node connectivity $\kappa(u, v)$ between all pairs of different nodes. Optionally store the result for each pair. Either use a flow-based algorithm (exact but slow) or White and Newman's approximation for local node connectivity (approximate but a lot faster).

2. Build an auxiliary graph H with all nodes in this bicomponent of G_{k-core} with edges between two nodes if $\kappa(u, v) \geq k$. For each biconnected component of H:

3. Compute the core number of each node in $H_{bicomponent}$, sort the values in reverse order (biggest first), and for each value c_{value}:

    a) Build a subgraph $H_{candidate}$ induced by nodes with core number *exactly* equal to c_{value} plus nodes in H that are conected with all nodes with core number equal to c_{value}.

        i. If $H_{candidate}$ has order $c_{value} + 1$ then it is a clique and all nodes will have degree $n - 1$. Build a core subgraph $G_{candidate}$ of level k of G induced by all nodes in $H_{candidate}$ that have core number $\geq k$ in G.

        ii. If this is not the case, then some nodes will have degree $< n - 1$. Remove all nodes with minimum degree from $H_{candidate}$. Build a core subgraph $G_{candidate}$ of level k of G induced by the remaining nodes of $H_{candidate}$ that have core number $\geq k$ in G.

            A. If the resultant graph is trivial or empty, continue with the following candidate.

            B. Else recompute the core number for each node in the new $H_{candidate}$ and go to (i).

    b) The nodes of each biconnected component of $G_{candidate}$ are assumed to be a k-component of the input graph if the number of nodes is greater than k.

    c) Compute the average connectivity of each detected k-component. Either use the value of $\kappa(u, v)$ computed in step 1 or recalcualte $\kappa(u, v)$ in the induced subgraph of candidate nodes.

Notice that because our approach is based on computing node independent paths between pairs of different nodes, we are able to use these computations to calculate both the cohesive

structure and the average node connectivity of each detected k-component. Of course, computing average connectivity comes with a cost: either more space to store $\kappa(u, v)$ in step 1, or more computation time in step 3.c if we did not store $\kappa(u, v)$. This is not possible when applying the exact algorithm for k-components proposed by Moody and White (2003) because it is based on repeatedly finding k-cutsets and removing them, thus it does not consider node independent paths at all.

The output of these heuristics is an approximation to k-components based on extra-cohesive blocks. We find extra-cohesive blocks and not k-components because we only build the auxiliary graph H one time on each bicoennected component of a core subgraph of level k from the input graph G. Local node connectivity is computed in a subgraph that might be larger than the final $G_{candidate}$ and thus some node independent paths that shouldn't could end up being counted.

Accuracy can be improved by rebuilding H from the pairwise node connectivity in $G_{candidate}$ and following the remaining steps of the heuristics at the cost of slowing down the computation. There is a trade-off between speed and accuracy. After some tests we decided to compute H only once and lean towards the speed pole of the trade-off. Our goal is to have an usable procedure for analyzing networks of thousands of nodes and edges in which we have substantive interests. Following this goal, the use of White and Newman (2001) approximation algorithm for local node connectivity in step 3.b is key. It is almost on order of magnitude faster than the exact flow-based algorithms. As usual, speed comes with a cost in accuracy: White and Newman (2001) algorithm provides a strict lower bound for the local node connectivity. Thus, by using it we can miss an edge in H that should be there. Therefore, a node belonging to a k-component could be excluded by the algorithm if we use White and Newman (2001) approximation in step 3.b . This is a source of false negatives in the process of approximating the k-component structure of a network. However, as we discussed above, the inaccuracy of this algorithm for sparse networks in reduced because in those networks the probability that a short node independent path uses nodes that could belong to two or more longer node independent paths is low.

Our tests reveal that the use of White and Newman (2001) approximation does indeed underestimate the order of some k-components, particularly in not very sparse networks. One approach to mitigate this problem is to relax the strict cohesion requirement of $H_{candidate}$ being a clique. Following the network literature on cliques, we can relax its cohesion requirements in terms of degree, coreness and density. We did some experiments and found that a good relaxation criteria is to set a density threshold of 0.95 for $H_{candidate}$; it doesn't increase false positives and does decrease the false negatives derived from the underestimation of local node connectivity of White and Newman (2001) algorithm. Other possible criteria that has given good results in our tests is permitting a variation in degree of 2 in $H_{candidate}$ —that is, that the absolute difference of the maximum an the minimum degree in $H_{candidate}$ is at most 2. The former relaxation criteria is used for all analysis presented below and in the appendix.

This algorithm can be easily generalized so as to be applicable to directed networks provided that the implementation of White and Newman's approximation for pairwise node independent paths supports directed paths (which is the case in our implementation of this algorithm on top of NetworkX library). The only change needed then is to use strongly connected components instead of bicomponents. And, in step 3, to start with core number 2 instead of 3.

In appendix B.1 we present an illustration of the heuristics using a convenient small synthetic network. In appendix B.2 we present an analysis of the performance of the heuristics compared to the performance of the exact algorithm for finding k-components (Moody and White, 2003). In appendix B.3 we discuss the implementation details of the heuristics; and in appendix B.4 we present the python code of our implementation of the heuristics for illustrative purposes[3].

3.5 Structural cohesion in collaboration networks

The structural cohesion model can be used to explain cooperation in different kinds of collaboration networks; for instance, coauthorship networks (Moody, 2004; White et al., 2004) and collaboration among biotech firms (Powell et al., 2005). Most collaboration networks are bipartite because the collaboration of individuals has as a result —or, at least, as a relevant byproduct— some kind of object or event to which its authors are related. All these papers follow the usual practice to deal with two-mode networks: focus the analysis only on one-mode projections. As such, we don't know how much information about their cohesive structure we lose by ignoring the underlying bipartite networks. Recent literature on two-mode networks strongly suggests that it is necessary to analyze two-mode networks directly to get an accurate picture of their structure. For instance, in small world networks, we do know that focusing only on projections overestimates the smallworldiness of the network (Uzzi et al., 2007). We also know that generalizing clustering coefficients to bipartite networks can offer key information that is lost in the projection (Robins and Alexander, 2004; Lind et al., 2005; Opsahl, 2011). Finally, the loss of information is also critical in many other common network measures: degree distributions, density, and assortativity (Latapy et al., 2008). We show that this is also the case for the k-component structure of collaboration networks.

Structural cohesion analysis based on the k-component structure of bipartite networks has been conducted very rarely and only on very small networks (White et al., 2004). The limited diffusion of these studies can be readily explained by the fact that bipartite networks are usually quite a lot bigger than their one-mode counterparts, and the computational requirements, once again, stifled empirical research in this direction. Other measures have been developed to deal with cohesion in large bipartite networks, such as (p, q)-cores or 4-ring islands (Ahmed et al., 2007). However, the former is a bipartite version of k-cores and thus it has the same limitations for subgroup identification; while the latter is very useful to determine subgraphs in large networks that are more strongly connected internally than with the rest of the network, but also lacks some of the key elements of the definition for groups in the sociological literature, such as being hierarchical and allowing for overlaps.

The heuristics for structural cohesion presented here allows us to compute connectivity-based measures on large networks (up to tens of thousands of nodes and edges) quickly enough to be able to build suitable null models. Furthermore we will be able to compare the results for bipartite networks with their one-mode projections. To illustrate those points we use data on collaboration among software developers in one organization (the Debian project) and scientists publishing papers in the arXiv.org electronic repository in two different scientific fields:

[3]The fully functional Python code is available from the authors

	Bipartite				Unipartite			
Network	# nodes	# edges	Av. degree	Time(s)	# nodes	# edges	Av. degree	Time(s)
Debian Lenny	13121	20220	3.08	1105.2	1383	5216	7.54	204.7
High Energy (theory)	26590	37566	2.81	3105.7	9767	19331	3.97	7136.0
Nuclear Theory	10371	15969	3.08	1205.2	4827	14488	6.00	3934.1

Table 3.2: Collaboration networks analyzed from science and from software development. See text for details on their content. Time refers to the execution of our heuristics on each network expressed in seconds.

High Energy Theory and Nuclear Theory. We built the Debian collaboration network by linking each software developer with the packages (i.e. programs) that she uploaded to the package repository of the Debian Operating System during a complete release cycle. We analyze the Debian Operating System version 5.0, codenamed "Lenny", which was developed from April 8, 2007, to February 1, 2009. Scientific networks are built using all the papers uploaded to the arXiv.org preprint repository from January 1, 2006, to December 31, 2010, for two well established scientific fields: High Energy Physics Theory and Nuclear Theory. In these networks each author is linked to the papers that she has authored during the time period analyzed. One-mode projections are always on the human side: scientists linked together if they have coauthored a paper, and developers linked together if they have worked on the same program. Table 3.2 presents some details on those networks.

In the remaining part of this section we perform three kinds of analysis to demonstrate the loss of information we incur when focusing only on one-mode projections when dealing with bipartite networks. First, we present a tree representation of the k-component structure —the cohesive blocks structure (White and Harary, 2001; Moody and White, 2003; White et al., 2004; Mani and Moody, 2014)— for our bipartite networks and their one-mode projections, both for actual networks and for their random counterparts. Second, we present a comparison among actual and random networks (both for one and two-mode) on the k-number frequencies of nodes. Finally, we present a novel graphic representation of the structural cohesion of a network, based on three-dimensional scatter plot, using average node connectivity as a synthetic and more informative measure of cohesion of each k-component.

For the first two analyses we do need to generate null models in order to discount the possibility that the observed structure of actual networks is just the result of randomly mixing papers and scientists or packages and developers. The null models used in this chapter are based on a bipartite configuration model (Newman, 2003), which consists of generating networks by randomly assigning papers/programs to scientists/developers but maintaining constant the distribution of papers per scientists and scientists by paper observed in the actual networks, that is the bipartite degree distribution. For one-mode projections, we generated bipartite random networks based on their original bipartite degree distribution, and then performed the one-mode projection. This is a common technique for avoiding overestimating the local clustering of one-mode projections (Uzzi et al., 2007). As the configuration model can generate some multiple edges and self-loops, we followed the usual practice of deleting them before the analysis in order to guarantee that random networks are simple, like actual networks.

So let's start with the tree representation of the cohesive blocks structure. As proposed by White et al. (2004), we can represent the k-component structure of a network by drawing a tree whose nodes are k-components; two nodes are linked if the k-component of higher level is nested inside the k-component of lower level (see Mani and Moody (2014, 1643,1651) for this kind of analysis on the Indian interorganizational ownership network). This representation of the connectivity structure can be built during the run time of the exact algorithm. However, because our heuristics are based on finding node independent paths, we have to compute first the k-components hierarchy, and then construct the tree that represents the connectivity structure of the network.

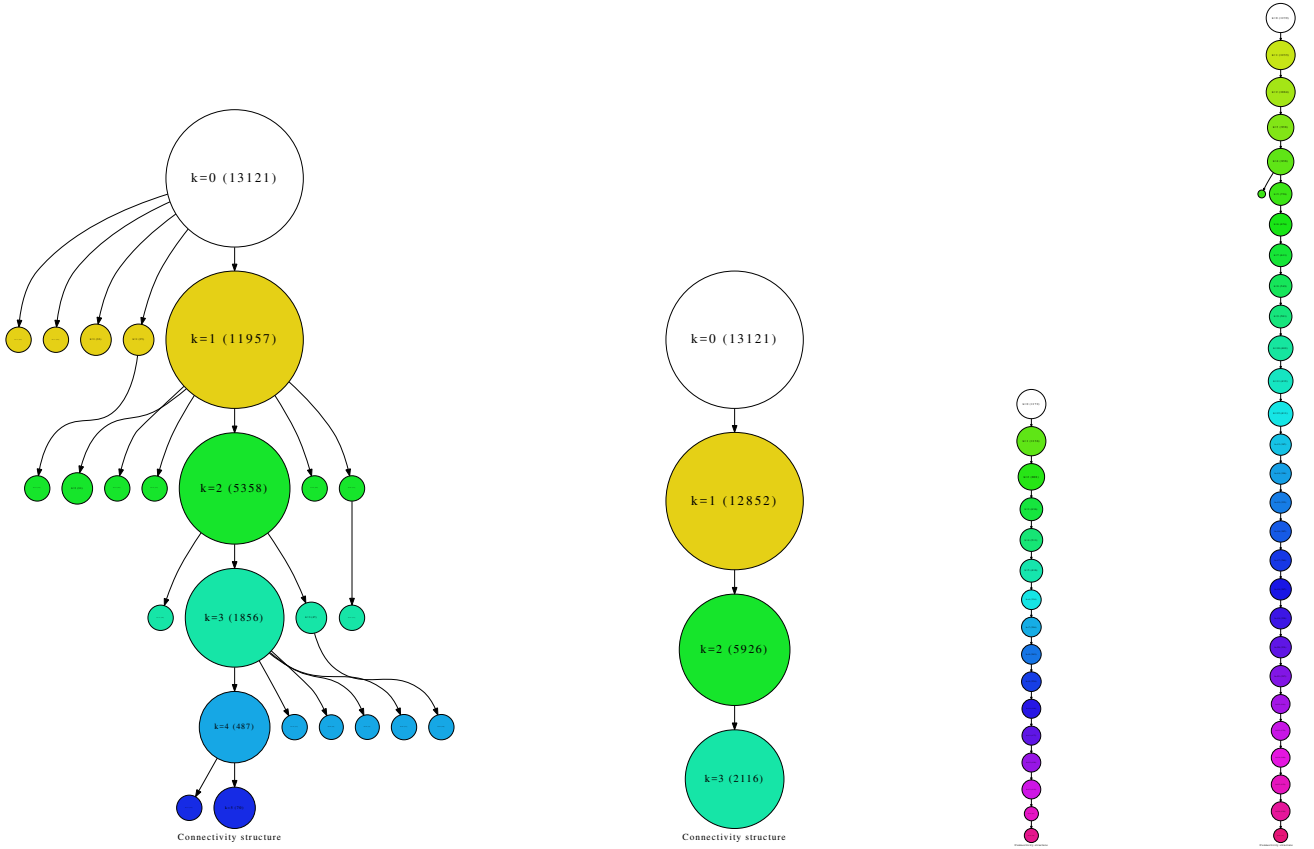
(a) Actual 2 mode

(b) Random 2 mode

(c) Actual 1 mode

(d) Random 1 mode

Figure 3.1: Cohesive blocks for two-mode and one-mode Nuclear Theory collaboration networks, and for their random counterparts. Random networks were generated using a bipartite configuration model. We built 1000 random networks and chose one randomly, see text for details. For lower connectivity levels we have removed some small k-components to improve the readability: we do not show 1-components with less than 20 nodes, 2-components with less than 15 nodes, or tricomponents with less than 10 nodes.



(a) 2 mode (b) Random 2 mode (c) 1 mode (d) Random 1 mode

Figure 3.2: Cohesive blocks for two-mode and one-mode Debian collaboration networks, and for their random counterparts. Random networks were generated using a bipartite configuration model. We built 1000 random networks and chose one randomly, see text for details. For lower connectivity levels we have removed some small k-components to improve the readability: we do not show 1-components with less than 20 nodes, 2-components with less than 15 nodes, or tricomponents with fewer than 10 nodes.

Figures 3.1a and 3.1c show the connectivity structure of Nuclear Theory collaboration networks represented as a tree, the former for the two-mode network and the latter for one-mode ones. As we can see, both networks display non-trivial structure. The two-mode network has up to an 8-component, but most nodes are in k-components with $k < 6$. Up to $k = 3$ most nodes are in giant k-components, but for $k = \{4, 5\}$ there are many k-components of similar order. Figure 3.1c, which corresponds to the one-mode projection, has a lot more connectivity levels —a byproduct of the mathematical transformation from two-mode to one-mode. In this network, the maximum connectivity level is 46; the four long legs of the plot correspond to 4 cliques with 47, 31, 27 and 25 nodes. Notice that each one of these 4 cliques are already a separated k-component at $k = 7$. It is at this level of connectivity ($k = \{7, 8\}$) where the giant k-components start to dissolve and many smaller k-components emerge.

In order to be able to assess the significance of the results obtained, we have to compare the connectivity structure of actual networks with the connectivity structure of a random network that maintains the observed bipartite degree distribution. In this case, we compare actual networks with only one random network. We obtained it by generating 1000 random networks and choosing one randomly. Figures 3.1b and 3.1d show the connectivity structure of the random counterparts for Nuclear Theory collaboration networks. For the two-mode network, instead of the differentiated connectivity structure displayed by the actual bipartite network, there is a flatter connectivity structure, where the higher level k-component is a tricomponent. Moreover, instead of many small k-components at high connectivity levels, the random bipartite network has only giant k-components where all nodes with component number k are. In this case, the one-mode network is also quite different from its random counterpart. There are only giant k-components up until $k = 15$, where the four cliques observed in the actual network separate from each other to form distinct k-components.

The hierarchy of the connectivity structure displayed in these plots allows us to do meaningful comparisons between networks in terms of their connectivity structure. For instance, figures 3.2a and 3.2c show the connectivity structure of Debian collaboration networks. The former displays the bipartite connectivity structure, which is quite different from two-mode Nuclear Theory structure discussed above. Although there are some small k-components for each connectivity level, most of the nodes with k-number k are in a giant k-component that encompasses most of the nodes of that level. Even at the top level of connectivity ($k = 5$), 80 percent of the 88 nodes with k-number 5 are in the same 5-component. Figure 3.2c displays the cohesive block structure for its one-mode projection. It consists of a monotonous linear succession of increasingly smaller k-components nested inside each other.

Figures 3.2b and 3.2d show the connectivity structure of the random counterparts of Debian collaboration networks. The random one-mode projection has the same structure than its actual counterpart, a single long chain of k-components nested inside each other. However, the random two-mode structure is quite different from its actual counterpart: it consists of a chain of single cohesive blocks. At lower connectivity levels, up to $k = 3$, the random network have more nodes in those giant k-components than its actual counterpart; but the actual Debian two-mode network has a bigger 4-component and also 2 5-components that are not present in its random counterpart. Thus, in terms of their connectivity structure, two-mode networks are farther apart from their random counterparts than their one-mode projections.

Note that, so far, the comparison of actual networks with their random counterparts has

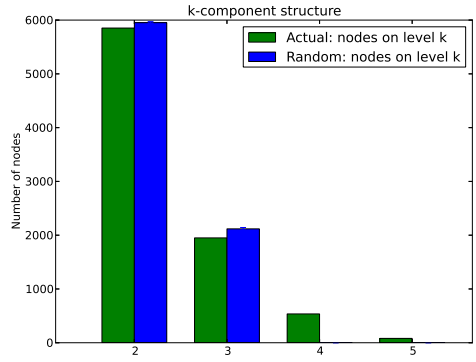
focused on a single random network. But, a single random network is not a sound null model. We do need to generate a large enough set of them and perform the connectivity analysis to have an accurate picture of possible connectivity structures generated solely by chance given the observed bipartite degree distribution. A good way to evaluate the differences between the actual network and the set of random networks is comparing the frequencies of k-numbers of their nodes. A node's k-number, or component number, is the value k of the highest order k-component in which it is embedded. In the barplots displayed in figure 3.3, each bar represents the number of nodes that have k-number k. Green bars represent k-number frequencies for the actual networks and blue bars represent the average value of 64 random networks that maintain the degree distribution of the original two-mode network. We analyzed 64 random networks to keep computation time reasonable, but we generated ten times more random networks and we have randomly selected one of each ten to perform the actual analysis.

Figure 3.3 shows that two-mode and one-mode projections of the same network yield quite different results in terms of k-number distribution among nodes when compared with their random counterparts. Bipartite collaboration networks have slightly fewer nodes with low component number (2 and sometimes 3) than their random counterparts. However, they have a lot more nodes in higher levels of connectivity. This means that, in bipartite random networks, the edges are more evenly distributed among all nodes. Thus more nodes are embedded in bicomponents, and in some cases, tricomponents; but also for this same reason, random networks have a lot fewer nodes in k-components of higher order (4, 5 or 6) than actual networks. Therefore, we can conclude that bipartite collaboration networks are significantly more hierarchical in connectivity terms than their random counterparts. As this hierarchy cannot be explained in terms of random mixing papers/programs with scientists/developers, it must be the result of an underlying organization principle that shapes the structure of these collaboration networks.

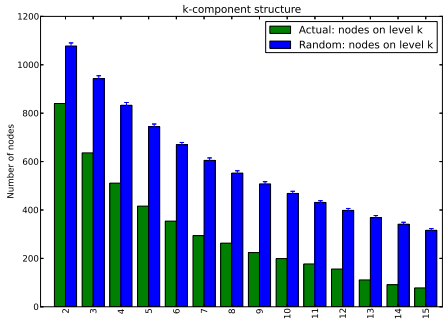
Going one step beyond classical structural cohesion analysis, as proposed above, we can deepen our analysis by also considering the average connectivity of the k-components of these networks. By analogy with the k-component number of each node, which is the maximum value k of the deepest k-component in which that node is embedded, we can establish the average k-component number of each node as the value of average connectivity of the deepest k-component in which that node is embedded. Notice that, unlike plain node connectivity, average node connectivity is a continuous measure of cohesion. Thus it provides a more granular measure of cohesion because we can rank k-components with the same k according to their average node connectivity.

Figure 3.4 graphically represents the three networks with three-dimensional scatter plots[4]. In these graphs, each dot corresponds to a node of the network, for two-mode networks nodes represent both scientists/developers and papers/programs. The Z axis (the vertical one) is the average k-component number of each node, and the X and Y axis are the result of a 2 dimensional force-based layout algorithm implemented by the `neato` program of Graphviz (Ellson et al., 2002). The two dimensional layout is computed by constructing a virtual physical model and then using an iterative solver procedure to obtain a low-energy configuration. Following Kamada and Kawai (1989), an ideal spring is placed between each pair of nodes

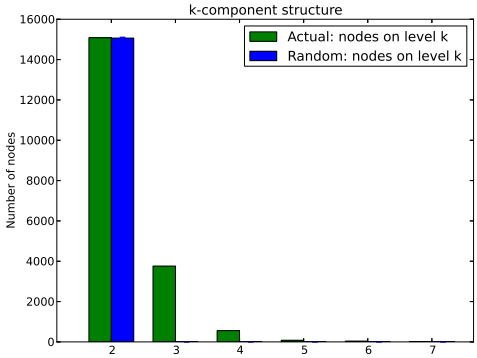
[4]These plots are produced with the powerful Matplotlib python library (Hunter, 2007).



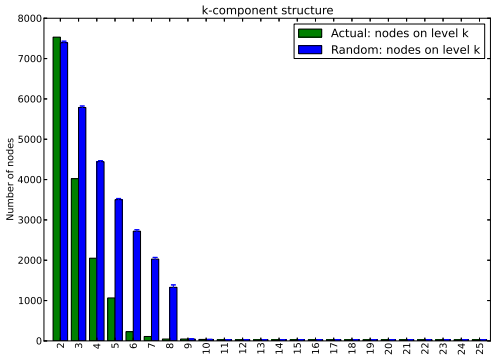
(a) Bipartite network formed by developers and packages during 2 years of collaboration (from 2007 to 2009) on the release codenamed Lenny of the Debian operating system



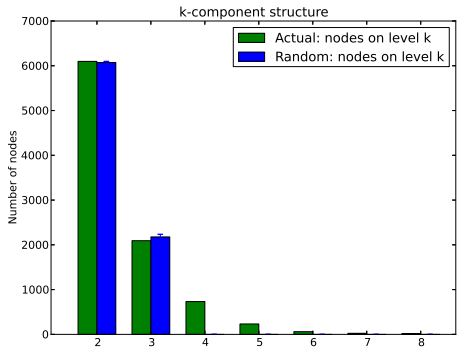
(b) Unipartite network formed by developers during 2 years of collaboration (from 2007 to 2009) on the release codenamed Lenny of the Debian operating system



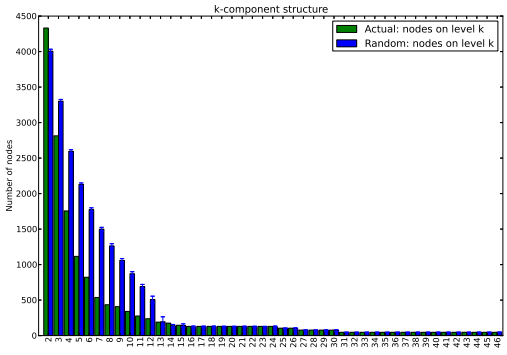
(c) Bipartite network formed by scientists and preprints during 5 years (2006-2010) in the high energy physics (theory) section of arXiv.org



(d) Unipartite network formed by scientists during 5 years (2006-2010) in the high energy physics (theory) section of arXiv.org



(e) Bipartite network formed by scientists and preprints during 5 years (2006-2010) in the nuclear physics (theory) section of arXiv.org



(f) Unipartite network formed by scientists during 5 years (2006-2010) in the nuclear theory section of arXiv.org

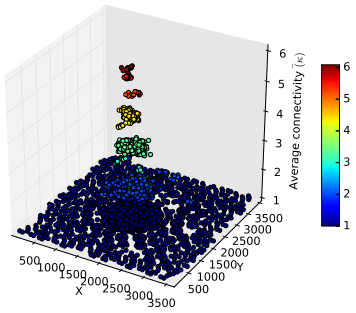
Figure 3.3: Barplots of k-number frequencies for two-mode and one-mode collaboration networks and their random counterparts. Green bars represent the actual k-number frequencies and blue bars represent the average k-number frequencies for 64 random networks that maintain the degree distribution of the original two-mode network.

(even if they are not connected in the network). The length of each spring corresponds to the geodesic distance between the pair of nodes that it links. The final node positioning in the layout approximates the path distance among pairs of nodes in the network.

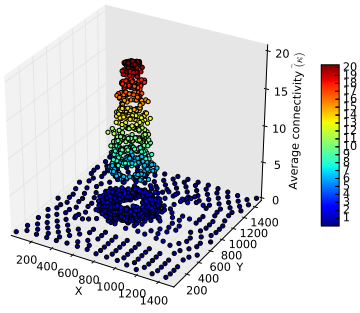
This novel graphic representation of cohesion structure is inspired by the approximation technique developed by Moody (2004) for plotting the approximate cohesion contour of large networks to which is not practical to apply Moody and White's exact algorithm for k-components 2003. Moody's technique is based on the fact that force-based layouts algorithms tend to draw nodes within highly cohesive subgroups near each other. Then we have to divide the surface of the two-dimensional plane in squares of equal areas and compute node independent paths on a sample of pairs of nodes inside each square so as to obtain an approximation for the node connectivity in that square. Then we can draw a surface plot using a smoothing probability density function. However, in order to obtain a nice smooth surface plot, we have to use heavy smoothing in the probability density function, and carefully choose the area of the squares (mostly by trial and error). Moreover, this technique strongly relies on the force-based layout algorithm to put nodes in highly cohesive subgroups near each other —something which is not guaranteed because they are usually based in path distance and not directly on node connectivity. Because we are able to compute the k-component structure with our heuristics for large networks, the three-dimensional scatter plot only relies on the layout algorithm for setting the X and Y positions of the nodes, while the Z position (average node connectivity) is computed directly from the network. Moreover, we don't have to use a smoothed surface plot because we have a value of average connectivity for each node, and thus we can plot each node as a dot on the plot. This gives a more accurate picture of the actual cohesive structure of a network.

Our synthetic representation of their cohesive structures can help researchers visualize the presence of different organizational mechanisms in different kinds of collaboration networks. The difference between the Debian and the scientific collaboration networks is striking. In figure 3.4a we can see the scatter plot for a Debian bipartite network. We can observe a clear vertical separation among nodes in different connectivity levels. This is because almost all nodes in each connectivity level are in a giant k-component and thus they have the same average connectivity. In other words, developers in Debian show different levels of engagement and contribution, with a core group of developers deeply nested at the core of the community. This pattern is the result of formal and informal rules of collaboration that evolved over the years (O'Mahony and Ferraro, 2007a) into a homogeneous hierarchical structure, where there is only one core of highly productive individuals at the center. Not surprisingly, perhaps, the Debian project has been particularly resilient to developers' turnover and splintering factions.

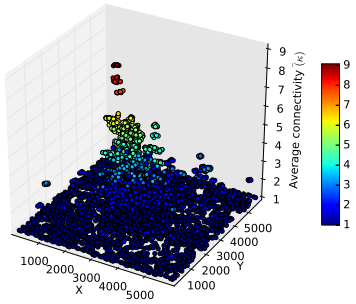
Scientific collaboration networks show a rather different structure of collaboration. The two-mode science collaboration networks (figures 3.4c and 3.4e) display a continuous hierarchical structure in which there are nodes at different levels of average connectivity for each discrete plain connectivity level. This is because science collaboration networks have a complex cohesive block structure where there are a lot of independent k-components in each plain connectivity level, for $k \geq 3$. Each small cohesive block has a different order, size and average connectivity; thus, when we display them in this three-dimensional scatter plot we observe a continuous hierarchical structure that contrasts with the almost discrete structure of Debian collaboration networks.



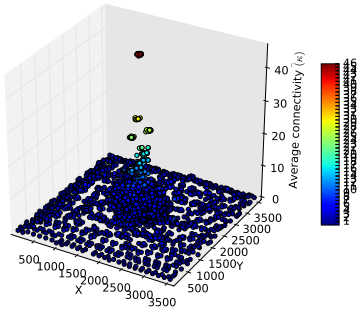
(a) Debian Lenny 2 mode



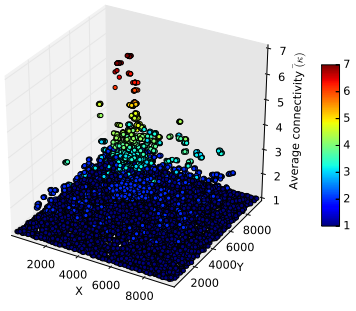
(b) Debian Lenny 1mode



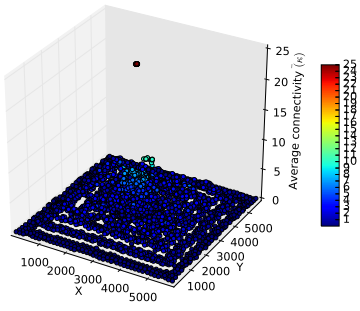
(c) Nuclear Theory 2 mode



(d) Nuclear Theory 1 mode



(e) High Energy Theory 2 mode



(f) High Energy Theory 1 mode

Figure 3.4: Average connectivity three-dimensional scatter plots. X and Y are the positions determined by the Kamada-Kawai layout algorithm. The vertical dimension is average connectivity. Each dot is a node of the network and two-mode networks contain both papers/programs and scientists/developers.

One explanation why we observe this heterogeneous connectivity structure is that scientific collaborations cluster around a variety of different aims, methods, projects, and institutional environments. Therefore as the most productive scientists collaborate with each other, hierarchies naturally emerge. However, we are less likely to observe one single hierarchical order as we did in the Debian network, as more than one core of highly productive scientists is likely to emerge. In a way our visualization captures the structure of the "invisible college" of the scientific discipline.

If we compare the bipartite networks with their one-mode projections using this graphical representation (see figures 3.4b, 3.4d, and 3.4f) we can see that, again, they look quite different. While bipartite average connectivity structure for the Debian network is characterized by clearly defined and almost discrete hierarchical levels, its one-mode counterpart shows a continuous hierarchical structure. However, this is not caused by the presence of many small k-components at the same level k, as in the case of bipartite science networks discussed above, but by the close succession of hierarchy levels with almost the same number of nodes in a chain-like structure (as depicted in figure 3.2c).

For collaboration science networks, the three-dimensional scatter plots of one-mode projections are also quite different than their original bipartite networks. They have a lot more hierarchy levels than bipartite networks but most nodes are at lower connectivity levels. Only a few nodes are at top levels of connectivity, and they all form part of some clique, which are the groups in the long "legs" of the cohesive block structure depicted in figure 3.1c. Thus, the complex hierarchical connectivity structure of bipartite collaboration networks gets blurred when we perform one-mode projection. An important consequence of the projection is that only a few nodes embedded in big cliques appear at top connectivity levels and all other nodes are way down in the connectivity structure. This could lead the risk of overestimating the importance of those nodes in big cliques and to underestimate the importance of nodes that, despite being at high levels of the bipartite connectivity structure, appear only at lower levels of the unipartite connectivity structure.

3.6 Summary of Contributions

This chapter contributes to our understanding of structural cohesion in a number of ways.

First, we extended theoretically the structural cohesion model by considering not only plain node connectivity, which is the minimum number of nodes that must be removed in order to disconnect a network, but also the average node connectivity of networks and its cohesive groups, which is the number of nodes that, on average, must be removed to disconnect an arbitrary pair of nodes in the network. Taking into account average connectivity allows a more granular conception of structural cohesion, and we show in our empirical analysis of collaboration networks how this approach leads to useful implications in empirical research.

Second, we developed heuristics to compute the k-components structure, along with the average node connectivity for each k-component, based on the fast approximation to compute node independent paths (White and Newman, 2001). These heuristics allow for the computing of the approximate value of group cohesion for moderately large networks, along with all the hierarchical structure of connectivity levels, in a reasonable time frame. We showed that these

heuristics can be applied to networks at least one order of magnitude bigger than the ones manageable by the exact algorithm proposed by Moody and White (2003). To ensure reproducibility and facilitate diffusion of these heuristics we provided a very detailed description of the implementation, along with an illustration of the source code [5].

Finally, we used the heuristics proposed here to analyze three large collaboration networks. With this analysis, we showed that the heuristics and the novel visualization technique for cohesive network structure help us capture important differences in the way collaboration is structured. Obviously a detailed analysis of the institutional and organizational structures in which the collaborative activity took place is well beyond the scope and aims of this paper. But future research could leverage the tools we provide to systematically measure those structures. For instance, sociologists of science often compare scientific disciplines in terms of their collaborative structures (Moody, 2004) and their level of controversies (Shwed and Bearman, 2010). The measures and the visualization technique we proposed could nicely capture these features and compare them across scientific disciplines. This would make it possible to further our understanding of the social structure of science, and its impact in terms of productivity, novelty and impact. Social network researchers interested in organizational robustness would also benefit from leveraging the structural cohesion measures to detect sub-groups that are more critical to the organization's resilience, and thus prevent factionalization. Exploring the consequences of different forms of cohesive structures will eventually help us further our theoretical understanding of collaboration and the role that cohesive groups play in linking micro-level dynamics with macro-level social structures.

[5] We believe that providing detailed implementation is critical to ensure reproducibility, but often these details are black-boxed, some times because of proprietary software restrictions or authors' reluctance to share their work.

Part III

Empirical analysis

4

Brief historical outline: from UNIX and C to Debian and Python

The software is the logic part of a computer system, while the hardware is the physical part. Loosely speaking we could say that computer programs are written in different programming languages, the set of instructions forming a program is called source code. This source code is compiled in order to translate it to a language that can be executed by the hardware; the result is the binary code —a sequence of 0 and 1—. Based only in the binary code of a program no human can understand how the program develops the task for which it was designed. It is like a black box that receives some *inputs* and returns some *outputs*.

There are two types of software programs according to their function within a computer system: application software are programs that develop specific tasks useful to the user, such as a word processor or a web browser. System software are programs that conform the operating system, according to Tanenbaum and Woodhull the two main functions of an operating system are, on the one hand, being an abstraction layer that provides to applications — and to developers who write applications— a set of simple operations that hide the complexity of hardware. On the other hand, the operating system is responsible for managing system resources —RAM, processor, disk space ,...— between different programs competing for them when they run in the system (Tanenbaum and Woodhull, 1996, 3-5). From a more practical but less rigorous point of view, we can conceive an operating system as the minimal set of programs that allow a computer to do useful things. Because what is considered useful has evolved along time, operating systems have also evolved.

At the beginnings of computer science, there was no clear distinction between software and hardware and between developer and user. There were only people who gave precise instructions to computers on which they should do. In 1952 IBM commercialized the first computer and during this decade began to spread their use (Weber, 2004, 21). One of the key events that marked the evolution of software at this time was the decision of the US Department of Justice that Western Electric and American Telephone and Telegraph (AT&T) could not join to work together beyond the field telecommunications. This decision, taken in 1956 under anti-trust legislation, lead AT&T to promote software licenses to a nominal cost and to put in the public domain the output of research developed in Bell Labs, in order to not violate anti-trust laws (Roca, 2007, 20).

4.1 UNIX and the C language

In the 1960s, computers were very expensive facilities that were only available to government centers, some large companies and universities. The study, design and implementation of operating systems largely focused research efforts in those years. A major project was the result of collaboration between MIT researchers and staff of Bell Labs, the goal was to build an operating system called MULTICS (*Multiplexed Information and Computing Service*. The results of those efforts did not success; in 1969 AT&T withdrew from the project. But two researchers who had participated in the work on MULTICS, Ken Thompson and Dennis Ritchie, developed on their own a new operating system, based in part on their work in MULTICS, which was called UNIX. The first implementation was made entirely by Thompson in a month during the summer of 1969 and consisted in a *kernel*, a shell, a text editor and an assembly language (Weber, 2004, 26).

Assembly languages are tightly linked to each hardware plataform, that is, each kind of computer has its own assembly language, which is incompatible with other assembly languages. Thus it was not possible to run software written for a computer in any other computer. In the early 1970s, Dennis Ritchie invented C, a general-purpose programming language, which allowed to write the source code of software once, and were able to run in a wide range of hardware thanks to a compiler, that translated the common C source code to the particular hardware instructions for each kind of computer. Until the early 1980s, although compilers existed for a variety of hardware, the language was almost exclusively associated with UNIX; more recently, its use has spread much more widely, and today it is among the programming languages most commonly used (Ritchie, 1993).

The first impulse to the spread of UNIX operating system was a computer science symposium in which Thompson and Ritchie presented a paper about UNIX. The authors offered to send a copy of UNIX to whom were interested, petitions exceeded by far initial expectations of the authors. The interest grew when they rewrote the UNIX with the C programming language, so it could work on any hardware that had a C compiler. Given this growing interest, AT&T —constained by anti-trust legislation— decided to license UNIX, at first, under minimal conditions: the software was provided without warranty (*'as-is'*) and without support or correction of errors by the company; paying a fee of several hundred of dollars AT&T sent a copy of the source code of UNIX (Weber, 2004, 28-29).

The UNIX operating System became popular as a teaching and research tool in computer science departments in universities around the world, especially at US. The availability of source code enabled experimentation, modification and improvement of the UNIX system. AT&T did not offer support or maintenance, so system users had a strong incentive to share solutions to bugs and improvements with the user community. Users of computers at that time were not like today, were in large part, college students, scientists or engineers with extensive technical training. UNIX was one of the first experiences of collaboration and knowledge exchange on a large scale in the field of software.

One of the main actors in the development of UNIX was the University of Berkeley, which began its own distribution of UNIX called BSD (*Berkeley Software Distribution*) in the 1970s. This distribution was started based on the UNIX source code of the company AT&T, but they added significant improvements. In 1976 Thompson joined the Berkeley team. The

authorship of the contributions to the source code were collected in the same source, following the practice established by Thompson and Ritchie (Weber, 2004, 27). In 1983 Berkeley's team published the 4.2 version of BSD UNIX which had major improvements, the most notable of which was an implementation of the TCP/IP stack —the communication protocol of Internet—, this version of BSD UNIX is one of the foundations of the Internet as we know it today (Weber, 2004, 35). The 4.2 version of BSD UNIX directly competed with a version of the company AT&T but it was far superior technically. The liberal license terms of BSD UNIX allow to build proprietary implementations on top of BSD UNIX, this allowed the emergence of new companies that commercialized modified versions of UNIX.

Requests for licenses from UNIX in the late seventies and early eighties increased considerably, mainly from large companies, military institutions, universities and research centers. AT&T and Bell Labs had to be separated by court order in 1984. As a result, Bell Labs began trading for the price of hundreds of thousands of dollars for new licenses of UNIX, which restricted drastically the number of institutions that could afford it. They also began a series of lawsuits in order to prevent the free dissemination of the various implementations of UNIX, particularly the implementation of Berkeley (Roca, 2007, 22). The dynamics of litigation lasted until the 1990s. This fact jeopardized the development of UNIX BSD; the future legal viability of the system was concerned. This uncertainty prompted the emergence of alternatives that, in its infancy, were technically inferior.

4.2 GNU and Linux

Richard Stallman started working in the Laboratory of Artificial Intelligence (AI) of MIT in 1971. In his own words, he joined a community that have shared the software for many years. Stallman says poetically that the act of sharing software is as old as computers, just as sharing recipes is as old as cooking (Stallman, 1998). At that time, the source code was accessible to all users and the act of sharing modifications involving improvements with the rest of the community was the norm. According to Stallman, this situation changed in the early eighties of the twentieth century when the community of MIT hackers collapsed. One spin-off of the MIT AI lab, hired almost all the people working there. The contract contained a non-disclosure agreement forcing people to not disclose their work and therefore prevented them to publish or share their work. In addition, in 1982, the MIT AI lab changed its hardware and a proprietary operating system were installed on them.

According to Stallman, those events led him to abandon his work at the MIT AI lab because, on the one hand, ethically he could not continue working with proprietary software and, on the other hand, the community in which he worked was dismantled. But rather than stop using software and engage in other activities, he decided to promote the construction of a new community within which they could restore the practice of sharing software. Stallman explained that he thought that the first step to restore the community was building a free operating system, because it is the essential tool in order to make a computer work. Thus was born the GNU project (*GNU is Not UNIX*) which aimed to create a general-purpose operating system that was a completely free reimplementation of UNIX (Stallman, 1985).

Stallman devise a set of formal rules, which revolve around the concept of copyleft. This

new concept is supported in the legislation about copyright. In Stallman's words, the idea of copyleft is that the author of a program gives everyone, without exception, permission to execute, copy, modify and distribute modified versions of the program. Stallman argues that in order to be effective, copyleft requires that derivative works of a program must also be free (Stallman, 1998). In this way privatization by software companies can be avoided, unlike free licenses that are permissive with private ownership, as the BSD license which allows a company to make changes to a free program and commercialize it in a binary format without providing any changes in source code form. The concrete implementation of these formal rules is the GNU/GPL License.

To articulate the process of building this new free operating system, Stallman founded in 1985 the Free Software Foundation (FSF), a nonprofit foundation with the objective of supporting the free software movement and give them legal cover. A relatively small group of people joined the efforts of Stallman, which was, in part, responsible for strategic planning in the early years of GNU. Hackers of the FSF created many free programs, some of them proved to be the best in their field. In the early nineties, the GNU project had a wide range of software but lacked a kernel —the program that interacts directly with the hardware— to have a complete operating system.

In this context, Andrew Tanenbaum created the first version of Minix in 1987. Minix is an operating system written from scratch by Tanenbaum and their students. The main objective was to allow his students to learn by analyzing how it is made and how it works an actual operating system. The Minix's source code is supplied as part of Tanenbaum and Woodhull (1996) book on operating systems. Linus Torvalds was a student at the University of Helsinki when he developed the first version of the Linux kernel. His aim was to write a new implementation of Minix for the popular and cheap i386 architecture. One of the key factors for the success of Linux was that Linux Torvalds decided to license it under the GNU/GPL license because the tools he used to develop Linux came from the GNU project. He released the source code on-line and asked everyone who wanted to collaborate with the project to submit improvement proposals.

Eric Raymond, a hacker from the old school, exposed the Linux development model in a work that has had a significant influence in the field of software development: The Cathedral & the Bazaar (Raymond, 1999). He starts by explaining the perplexity he felt when he became interested in Linux. Since the mid 80's had worked with the FSF by writing free software and always had a development model that, metaphorically, can be compared with building a cathedral. A small group of architects design the program, implement it and test it for a long time. When it successfully pass all the tests it is released. He was surprised both with Linux and its development model, which consisted of releasing the program very often, although mistakes had not been solved. The main idea is relay on all the people who devote their free time to test and improve Linux, collect all the proposals and implement the best.

Raymond qualifies metaphorically this production model as a bazaar, where anyone can contribute code to the project and each project is responsible for integrating the proposals that seem useful to the source code of the program. It is fair to say that further studies on the actual dynamics of larger projects and relevant software —like Apache and Mozilla— have shown that the cathedral model is not entirely abandoned, rather the actual model is an hybrid; a combination of the two models where a significant portion of the program development is

provided by a relatively small group of people, but there are an extensive variety of people, who more or less sporadically, contribute to the project (Mockus et al., 2002).

We must consider that in the nineties starts a massive deployment of Internet in some countries. Internet is the infrastructure that makes possible to weave a network of peer collaboration that characterize this production model. This development model was not invented by Linus Torvalds, is not difficult to recognize the principle of peer review of scientific practice in it. In fact other software projects, such as BSD UNIX, had adopted a model that closely resembles the practices of scientific communities. It must be said that it was quite difficult for a contribution that came from an outsider of the Berkeley team was accepted in BSD UNIX; in this sense, the classical model of development of UNIX established at Berkeley was more elitist than the Linux model, which was more open and transparent but less rigorous. The key to understand the success of the Linux model is that it was contemporary to the spread of access to global digital networks in some countries.

In the early nineties, the GNU system was almost complete, just lacked the kernel; the gap was important because the kernel is the software that allows the system to operate autonomously on the hardware. Linux filled the void that was missing, the sum of the Linux kernel and GNU applications resulted in a general purpose operating system completely free: the GNU/Linux system. But the fact that all the pieces of the operating system were available did not meant that putting them to work together was an easy thing. In the first half of the nineties, installing a GNU/Linux system required a great deal of expertise and considerable time to devote to it. In this context appear and develop different distributions of the GNU/Linux operating system, among which is the Debian project, which is the subject of empirical analysis of this research.

4.3 The Debian Project

In the early nineties of the twentieth century the most powerful free operating system was BSD UNIX. But as we said, the litigation that was submitted by the companies who hold the copyrights of UNIX threatened its future viability. This led to the emergence of alternatives, although initially were technically inferior, were substantially improved in the late nineties and early twenty-first century. These alternatives were the GNU project and the Linux kernel, the combination of which allowed to build a completely free general purpose operating system. But combine these pieces of software was not, nor is, a trivial task. Linux was in its early stages of development, many people made contributions to the source code and new versions of Linux were released on a daily basis. Therefore, it was required a great effort to have GNU/Linux systems running, and even more, keep them updated. Especially for people who wanted to work *with* the GNU/Linux system to develop different tasks and not *in* the system (Krafft, 2005, 30).

In 1993 Ian Murdock, a student at Purdue University at Indiana took the initiative in creating the Debian Project[1] with the goal of building a distribution of GNU/Linux system. The Debian Project's initial proposals were included in the *Debian manifesto* (Murdock, 1994). Two of the main features of the Debian project are listed in the manifest. First, define a new

[1]The name Debian is the contraction of the names Debra —Murdock's wife— and Ian.

type of distribution of GNU/Linux, instead of being carried by a person or a closed group, the aim was develop the system following an open and decentralized mode inspired by Linux. Secondly, Debian was defined as a non-commercial project and focused on technical excellence instead of economic profits, but without sacrificing the aim to compete in excellence with commercial options, whether free or proprietary.

Murdock says, rightly, that the distributions are essential for the future of GNU/Linux system in order to eliminate the need for the user to search, download, compile, install and integrate a large number of programs that are the basic components of a functional system. Murdock notes that despite the importance of the distributions, they have not received much attention by free software developers. To maintain a well integrated, error-free and reasonably updated distribution of GNU/Linux system is not easy nor glamorous task, it requires a great amount of work and coordination to manage complexity. Many distributions of GNU/Linux system at the time —the most popular of which was Softlanding Linux System (SLS)— started with a technically acceptable level, but as time passed were degenerating because they did not solve the problems that arise nor updated versions of programs distributed. Thus, it was relatively easy to start a distribution of GNU/Linux system but it was very difficult to keep it operational and functional for significant periods of time.

The Debian project does not produce all the software that distributes; their main task is software integration. The source code of the software that composes the operating system is published originally under some kind of free license by authors that typically aren't involved in the project. The aim of Debian is to integrate useful programs and package them so that an average user —without deep knowledge of software engineering— can install or upgrade many programs in an easy and automated way. One of the main features of the production process of Debian is modularity, that is, dividing the project into semi-independent modules, designed to work together but that can be developed relatively independently. This feature allows people with different expertise, skills and motivation to participate in the development of the system at different levels and with different intensity.

Start date	Release date	Version	Code name	# developers	# source packages
1998-07-24	1999-03-09	2.1	Slink	275	1577
1999-03-09	2000-08-15	2.2	Potato	439	3465
2000-08-15	2002-07-19	3	Woody	896	6659
2002-07-19	2005-06-06	3.1	Sarge	1229	10733
2005-06-06	2007-04-08	4	Etch	1284	10300
2007-04-08	2009-02-14	5	Lenny	1426	11738
2009-02-14	2011-02-06	6	Squeeze	1618	13080
2011-02-07	2013-05-04	7	Wheezy	1754	14989

Table 4.1: Evolution of number of developers and packages in released stable version of the Debian operating system from 1998 to 2013. Data from the Ultimate Debian database: http://wiki.debian.org/UltimateDebianDatabase

Table 4.1 summarizes the evolution of the number of Debian developers and software packages when stable versions of the operating system were released. The availability of

data derived of the open nature of the project and the recent interest by community forms of organizing have triggered an interesting stream of research about the Debian project in the last years (O'Mahony, 2003; Coleman, 2005; O'Mahony and Ferraro, 2007b; Ferraro and O'Mahony, 2010). Those research efforts have focused mainly in the governance system, the membership process and the ethical motivations of developers. Thus we have a good understanding of the political and individual dynamics of the project but we lack a detailed analysis of its production related dynamics. Our aim is to fill this gap providing a longitudinal analysis of the global patterns of relations among developers in the production process. This allowed us to illustrate the relevance of the network approach in order to understand the development of an actual collaborative community.

4.4 The Python Language

In the late 1980s, Guido van Rossum —a dutch computer scientist working at the *Centrum Wiskunde & Informatica*[2]— invented the Python programming language. The CPython project produces the reference implementation of this free computer language. In the begining, it started as an individual effort of Guido van Rossum, and has become one of the mainstream computer languages in the XXI century. Until 2000 it was almost an individual effort of van Rossum with few close collaborators. From 2000 the project gained popularity and several developers joined the project. In 2013, 97 individuals contributed at least one line of source code to Python. Nowadays, the Python language is widely used in several key areas of computing and software development. Some of the biggest websites of the WWW are powered by Python, such as youtube.com and reddit.com. Another main area where Python is very prominent is scientific computing. For instance, all the data coming from the big telescopes on —and around— our planet are processed using tools mostly written in Python.

[2]Which translates in English to National Research Institute for Mathematics and Computer Science

5

Empirical Analysis

5.1 Conceptualizing dynamic hierarchies

The analysis of the hierarchical structure of organizations has been a central topic on organizational research in the last decades. This analysis has been mainly static in the sense that the focus of interest has been, among others, the distinctions between formal hierarchies and informal patterns of relations (Krackhardt and Hanson, 1993; McFarland, 2001), the comparative analysis of the shape of the hierarchy (Blau and Scott, 1962; Blau, 1964), the impact of different kinds of hierarchical structures in the outcomes of the organizations' activities, the potential contradictions among the internal hierarchical structure of organization and its goals towards a more egalitarian society (Michels, 1915; Selznick, 1949), to cite only a few key issues.

Despite the huge among of work devoted to the analysis of hierarchy in organizations, the dynamic dimension of the hierarchy has received a lot less attention. The work on the dynamic dimension has focused on the evolution of hierarchical structures of organizations through time (Blau, 1969). There is however another possible definition of dynamic dimension in the analysis of hierarchy in organizations: the ratio of renewal of the individuals in the positions defined by that hierarchy. This important element of the dynamic dimension of hierarchy has been partially approached from the perspective of vacancy chains (White, 1970; Stewman and Konda, 1983; Padgett, 1990). However this approach has focused mostly on the career paths of individuals inside organizations instead of focusing on the pace of renewal of individuals in the hierarchical structure of organizations.

We propose that hierarchical structures can be classified in a continuum, the two extreme points of which are, on the one hand, a static hierarchy —where when an individual is appointed in a position of the hierarchy, this position is for life— and, on the other hand, a dynamic hierarchy —where the individuals occupying positions defined by the hierarchy have a very high pace of renewal. Notice that, in this context, the hierarchy can refer to both the formal and informal patterns of relations. An example of static hierarchy is the catholic church, where an appointment —even far from the top level— will typically last for life. On the other hand, dynamic hierarchies are a lot less common, especially before the last years of the twentieth century.

Since then we have witnessed the emergence of new organizational forms, mainly around Free and Open Source Software projects (FOSS). We propose that one of the central characteristics of these new organizational forms is precisely their high ratio of turnover in key hierarchical positions, both in the formal and informal internal organization. We do think that this dynamic dimension has not been taken into account in the analysis of those new organizational forms, and only by considering and analyzing it we can deepen our understanding, not only of the new emerging organizational forms, but also further our understanding of organizations and the challenges that they face.

The dynamism of FOSS hierarchies

Free and Open Source Software (FOSS) communities have attracted a lot of attention from researchers of different fields since the late nineties of the past century. The first academic accounts of this phenomenon were mainly descriptive; their main focus was to just describe the organization of FOSS communities, the individual motivations of the people that form these communities, and the quality of the products that they produced (Benkler, Shaw, and Hill, Benkler et al.). Most of the interest was derived from the fact that FOSS communities do not conform to the accounts of collective dynamics and individual motivations established by the dominant neoclassical economic theories.

The academic efforts took mainly two directions. On the one hand, some authors tried to reconcile the dynamics of FOSS communities with neoclassical economic accounts. This effort was mainly focused on the individual motivations of the participants in those communities. They tried to explain these motivations in terms of rational self-interested individuals, as prescribed by dominant economic theories. On the other hand, other authors saw the emergence of FOSS communities as a new organizational form that provided a more democratic way of enabling collective production without the constrains imposed by the markets and/or bureaucratic organizational forms (Benkler, 2002, 2006; Castells, 2013).

The later accounts of FOSS communities were initially uncritically celebratory of the phenomenon. They were heavily influenced by practitioners and advocates of the FOSS phenomenon which emphasized the technical superiority of the products developed by FOSS communities, while maintaining an ethical stand that valued more cooperation and reciprocity than competition and self-interest. One of the most influent early accounts from practitioners was Raymond (1999) that proposed, among other things, that the technical superiority of FOSS software products was due to the "Linus law", which states that "given enough eyeballs, all bugs are shallow", suggesting that given a large enough developer and user community that have access to the source code, all software errors (ie "bugs") will be detected quickly and the solution will be obvious at least to someone.

Thus "Linus law" suggests that FOSS communities are composed by a large set of individuals loosely organized with a very flat or inexistent hierarchy among them, and that all individuals might contribute more or less the same: a pair of eyes that should look at the source code in order to improve it. This somewhat naive account of the dynamics of FOSS production process was accepted uncritically by many academics that were sympathetic with the arguments of the FOSS practitioners. Some critical voices, coming mostly from Computer Science, challenged this claim with sound empirical arguments; for instance Glass (2002) cor-

rectly noted that if "Linus Law" was right then the number of bugs found in a software project should increase linearly with the number of people looking at their source code. No such thing have been proved empirically. Also, "Linus Law" not only treats each pair of eyes (ie individuals) as equally important, it also implicitly assumes that all bugs are similar, which is very implausible.

Other early empirical research, coming mostly from Computer Science, has pointed out that even in big, mature and widely used FOSS projects, only few of the participants account for the lion's share of the work done. For instance, Mockus et al. (2002) show that less that 20 developers of the Apache project[1] contributed more than 80% of the code base. This core of developers is embedded in a larger set of participants, that mainly help reporting and fixing errors, answering questions about the software in public forums, and writing documentation. Later empirical research has confirmed that the distribution of contributions in FOSS projects is right-skewed and heavy tailed, meaning that most participants make very small contributions, and only few individuals make almost all relevant contributions.

Recent empirical research on peer production projects (concretelly user edited wikis) has also shown that these projects exhibit deep contribution inequalities (Shaw and Hill, 2014). The authors suggest that these projects may conform to Michels (1915) "iron law of oligarchy" which states that organizations tend towards oligarchy as they grow, even if democracy and participation are part of the core goals of the organization. Therefore, there is ample empirical evidence that confirms that there is an important differentiation of roles and functions among participants on FOSS communities. This fact does not fit well with the picture of a flat hierarchy of peers portrayed by early accounts of the phenomenon.

We do think that the narrative of a flat hierarchy of peers was so successful because the formal organization of most FOSS projects is usually quite fuzzy, and very different of the formal structure of other kinds of organizations. However, the informal structure emerging from the patterns of collaboration among individuals in a FOSS project is quite hierarchical because reflects the fact that only few individuals are responsible for most contributions to the project. We propose that the way to advance our theoretical understanding of the FOSS phenomenon is by analyzing their social structure. The social structure of a community are the patterns of relations established among individual participants in the process of building the software packages (or any other product, such as on on-line encyclopedia) that they release. The public nature of FOSS communities implies that most of the data generated in the production process is available, and thus an important source of empirical data that we can use to test competing theoretical accounts of the phenomenon.

We found that the developers that contribute the most are in the higher levels of the connectivity structure of the project's collaboration networks. Moreover, by analyzing the composition of individuals on these key topological positions we are able to assess to which extend there is turn over of individuals at the top of the connectivity structure. Our analysis shows that the ratio of renewal of individuals at this structural position is quite fast, which characterizes FOSS communities as dynamic hierarchies. Thus, if we analyze cross-sectionally (ie in a concrete point of time) a FOSS project, a very small fraction of the participants are the ones

[1] Apache is one of the most successful FOSS projects, it's flagship product is the Apache web server which powers more than 50% of the web sites that form the WWW.

that actually do the lion's share of contributions, as previous empirical research has shown. However if we analyze the evolution of contributions longitudinally, we find that the persons that contribute the most change through time. This continuous renewal of the people that does most of the work —what we call dynamic hierarchy— is a key mechanism to explain how FOSS projects, which are mostly voluntary based, geographically distributed, and mostly operated from the Internet, can thrive and evolve to a point where they are key pieces of the infrastructure that enables the Internet and other essential Information technologies.

Our focus on the rotation of individuals at the top levels of the connectivity structure brings us to the issue of the robustness of the FOSS communities. From a pure network perspective, it is usual to analyze robustness by removing nodes and measuring how this affects the size of the giant connected component in the network (Albert et al., 2000). Nodes are removed following different mechanisms; either at random —to simulate failure— or removing nodes according to their degree —to simulate a deliberate attack. However these mechanisms are best suited for analyzing the robustness of physical networks, such as the Internet. They clearly fall short for analyzing the robustness of FOSS communities, because not random failures nor targeted attacks are the main mechanisms through which the persons that work on FOSS communities turn over.

Our approach here is to analyze the median active life of developers in a FOSS project as a better way of assessing the robustness of a FOSS community. We also apply the well established survival analysis techniques (Miller Jr, 2011) in order to describe and model the flux of people throughout the history of a FOSS community. We found that the position of an individual in the connectivity structure of the collaboration network also impacts significantly in the median life of a developer in the project.

5.2 Empirical Setting

In this paper we analyze the social structure of two big and mature FOSS projects: The Debian operating system, and the reference implementation of the Python computer language. We focus on the structural positions in which the most active contributors are, and the median life of individual contributions in the project. Our main empirical interest is about the volume of contribution of each individual to the project, and the role of contributions —as independent variable— in relevant elements of a FOSS project, such as the median active life of individual contributors to the project.

The Debian project combines the Linux kernel and GNU userland tools, along with many other free programs, in order to release a free implementation of the UNIX operating system[2]. The Debian project does not produce all the software that distributes: their main task is software integration. The aim of Debian is to integrate useful programs and package them so that an average user —without deep knowledge of software engineering— can install or upgrade many programs in an easy and automated way. A software package is a program, or a closely related set of programs, that can be maintained semi-independently but has a standardized interface that allows integration with the rest of the operating system. To maintain a package means to take the source code of a program released under a free license (called *upstream*),

[2]What is usually referred as a Linux operating system.

adapt its default behavior in order to match the specifications of the Debian operating system, package it in a standardized form, and upload it to the repository of packages that form the operating system.

We build the collaboration network of Debain project based on the Ultimate Debian Database (UDD)[3] (Nussbaum and Zacchiroli, 2010). The UDD contains information related to the work of each individual in the project which allow us to build the developers-packages affiliation network. One developer is linked to every package she has uploaded in the archive in a period of one year. Therefore, the result is a 2-mode network with developers —the actors— and packages —the groups— as the two types of nodes. Then we can obtain the 1-mode projection on the developers side: two developers are linked if they have uploaded different updates of the same package during the same year.

The CPython project produces the reference implementation of a free computer language —named Python—. It started in 1991 as an individual effort of Guido van Rossum, a Dutch software engineer, and has become one of the mainstream computer languages in the XXI century. Until 2000 it was almost an individual effort of van Rossum with few close collaborators. From 2000 the project gained popularity and several developers joined the project. In 2013, 97 individuals contributed at least one line of source code to Python. Nowadays, the Python language is widely used in several key areas of computing and software development. Some of the biggest websites of the WWW are powered by Python, such as youtube.com and reddit.com. Another main area where Python is very prominent is scientific computing. For instance, all the data coming from the big telescopes on —and around— our planet are processed using tools written in Python.

In the case of the Python project, we build its collaboration network based on their central repository of source code [4]. Each developer is linked to the source code files that she has edited, thus we model it as bipartite network, with developers and source code files as the two kinds of nodes. Each relation developer – file is weighted by the total number of lines that the developer added or deleted from that file. Thus, individual contributions to the CPython project can be measured as the number of lines of code added or deleted from a source code file in the official distribution of the reference implementation of the Python language. This measure of contribution can be safely treated as a continuous variable in regression modeling.

5.3 Methods

Our modeling strategy to capture the patterns of relations among developers in these two projects is to focus on the actual contributions of each developer to the project. We model collaboration networks as bipartite graphs, where the two sets of nodes are, on the one hand, human developers and, on the other hand, entities that conform the product that is released by the FOSS project. In the case of Debian, these entities are software packages, and in the case of Python, they are source code files. Note that the collaboration network is based on individual contribution but it not only captures the total amount of contribution that a given individual does, but also to which part of the project the contributions are focused, and who

[3] https://wiki.debian.org/UltimateDebianDatabase [accessed July 2014]
[4] http://hg.python.org/cpython/ [accessed July 2014]

else in the project is also working on the same entities. This is why we name these bipartite graphs collaboration networks.

This modeling approach captures mostly the informal patterns of relations that individuals establish when contributing to the project. FOSS projects have a wide range of formal organizational forms, and in this respect, they can be quite different. The definition of the leadership position in the two projects in which we focus this paper nicely capture these differences in formal organization: Debian has a very developed formal bureaucracy, the project elects its leader each year through a secret vote of all its members after a electoral campaign where the candidates discuss among them and try to gain supports; Python instead has its original author —Guido van Rossum— in a permanent position of leadership, the people in the project refer to him, and his position of leadership, as "Benevolent Dictator For Life" (BDFL).

Despite these differences in the formal organization, if we focus on the patterns of relations among developers in the productive process, what we call the collaboration network, we can analyze the contribution dynamics, analyze hierarchical positions defined by these patterns, assess the pace of renewal in these positions, and determine the impact in the median active life on a developer in a project of being in a concrete hierarchical position.

One of the challenges that we faced, that is both theoretical and methodological, is how to define cohesive groups in collaboration networks. There are many ways of defining a cohesive group given a collaboration network. Our aim was to define groups in collaboration networks in a way that is theoretically sound from a sociological point of view. Network science is nowadays quite interdisciplinary, and a lot of physicist have recently proposed a bunch of techniques, under the label of community detection algorithms (Fortunato, 2010), that determine groups in networks based on the patterns of relations among the entities of the network.

However, these techniques are suboptimal from a sociological theory point of view because the four key elements that a sociologically sound group classification should have are not present in most, if not all, most used community detection algorithms (Torrents and Ferraro, 2015). The four key dimensions are: robustness (the groups should not depend on only one or few individuals to be a group), overlap (persons usually are part of more than one cohesive group), positional dimension (some actors, because of their position in the global patterns of relations, obtain preferential access to information or resources that flow through the network), and hierarchy (collaboration networks have *hierarchical structure* in the sense that highly cohesive subgroups are nested inside less cohesive ones).

Our conclusion is that the structural cohesion model, developed by White, Moody and Harary (White and Harary, 2001; Moody and White, 2003), is the best model for analyzing group formation in collaborative networks in the context of our empirical research. This model is based on the graph theoretic measure of node connectivity, and defines cohesive groups as k-components, that is, groups of nodes in which k nodes have to be removed in oder to disconnect the group. K-components form the connectivity structure of the network, and aptly capture the central elements of a sociological definition of cohesive group (Torrents and Ferraro, 2015).

However, there are some important practical difficulties related to the computation of the measures that characterize the structural cohesion model. Their time complexity is super quadratic, approximately of the order of the forth power of the size of the input network.

This makes non practical the exact computation of the k-component structure in networks bigger than several hundreds of nodes. We use here some useful heuristics that allow to approximately compute the connectivity structure of large sparse networks in a reasonable time frame (Torrents and Ferraro, 2015).

Once we built the collaboration networks for the two projects, and determined their connectivity structure, we perform a descriptive analysis of the percentage of total contributions by connectivity level. This simple descriptive analysis shows that there is a strong correlation between the position of a developer in the connectivity structure of the collaboration network and her total amount of contribution to the project.

We then deepen our analysis by modeling individual contributions to the project using different regression models in order to asses the relation of the structural positions that individuals occupy with their level of contribution to the project. For the case of the Debian project, contributions are uploads of packages to the central repository of the project, thus contributions in this context have to be modeled as a discrete variable. For this case we used a negative binomial regression model to deal with the over-dispersed count data from the values of the discrete contributions variable.

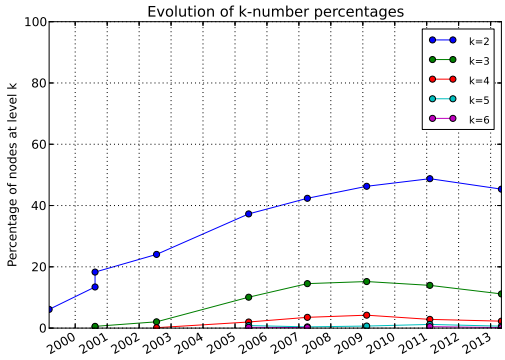
For the case of the Python project, contributions are lines of source code added or deleted from one of the source code files of Python's code base. We modeled contributions using a panel regression with individual fixed effects. This design allows us to account for unobserved variability among the individual developers, such as cultural background or coding expertise, and disentangle if the position of a developer in the connectivity hierarchy has an effect in her level of contribution to the project.

Finally, we are also interested in the impact of the position than an individual occupies in the collaboration network with her long term involvement with the project. To that end we applied Cox proportional-hazards regression for survival data to both Debian and Python projects. In its origin, survival analysis, was focused on modeling lifespans of individuals and is still widely used in medicine. However, this kind of analysis can also be used to model any kind of duration. We model the active life of a developer in a FOSS project to the period that this developer doing at least one contribution. We consider a developer "dead" when she no longer contributes.

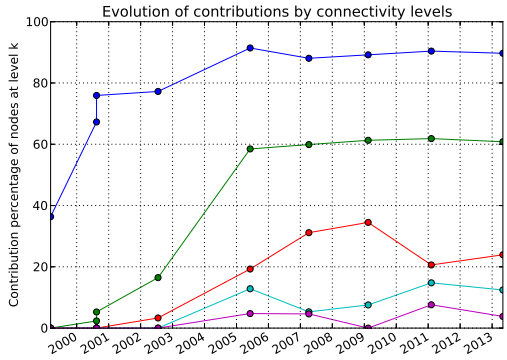
5.4 Results

Modeling individual contributions

As we discussed, the empirical work on FOSS communities has already established that it is only a small fraction of all participants in a project who are responsible for most contributions. As a first step for our analysis, we analyze the topological position of the individuals that contribute the most in the patterns of relations —the social structure— among individuals in that project. Following the structural cohesion model (Moody and White, 2003), we found that these individuals are part of the top connectivity levels of the collaboration network, that is, they are members of k-components of high k which represent cohesive subgroups nested inside each other in the collaboration network.



(a) Evolution of the percentage of nodes in each connectivity level for the Debian project.



(b) Evolution of the percentage of contributions by developers in each connectivity levels for the Debian project.

Figure 5.1: Evolution of the percentage of nodes in each connectivity level (left) and evolution of the percentage of contributions by developers in each connectivity levels (right) in the Debian project.

Figure 5.1a displays the evolution of the percentage of nodes in each connectivity level in the period under analysis. It is clear that, after some changes in Debian's productive process in 2004, there is a significant increment of the hierarchy of connectivity levels, and also an increment of nodes in top connectivity levels. The percentage of nodes in bicomponents, goes from less than 25% in 2002 to 38% in 2005 and peaks at almost 50% of the nodes at 2011. The percentage of nodes in tricomponents also experiments a sharp increment, it goes from 2% of the nodes in 2002 to 10% in 2005, and peaks around 15% in 2009. Components with higher connectivity levels only emerge after 2004, although these connectivity levels have a very small fraction of the nodes of the collaboration networks they play a critical role in terms of work done.

Figure 5.1b displays the percentage of contributions by developers in each connectivity level. We can see that, although there are very few developers in high connectivity levels, they are responsible for a big fraction of the total contribution in terms of packages uploaded to the Debian archive. For instance, 4-components contain from 4% to 8% of all developers in the network, but they did form 20% to 35% of all contributions. Tricomponents account from 15% to 25% of all developers, those developers are responsible for approximately 60% of all contributions. In the case of bicomponents, the developers in connectivity level ≥ 2 are responsible for approximately 90% of all contributions despite the fact that only between 50% and 60% of the developers are in components of $k \geq 2$.

Therefore, it is clear that there is a strong correlation between the connectivity level of a developer and her contribution to the project. To further the analysis, we modeled the contributions —which in this case are uploads of new versions of packages to the Debian archive— using a negative binomial regression. Which is well suited for the count nature of our dependent variable (# of uploads) and its over dispersion. We controlled the contributions of each developer, on the one hand, by several key variables related to the technical side of the production process, such as the size (*psizes*) and the dependencies of each package (*deps*), the bugs

reported (*bugs*), or the time that the developer has been active in the project (*tenure*). And, on the other hand, we also controlled for centrality measures including raw degree (*degree*) and closeness (*closeness*). As can be seen in the following table, the connectivity level in which a developer is embedded (*knum*) has a positive and significative impact on her contributions to the project.

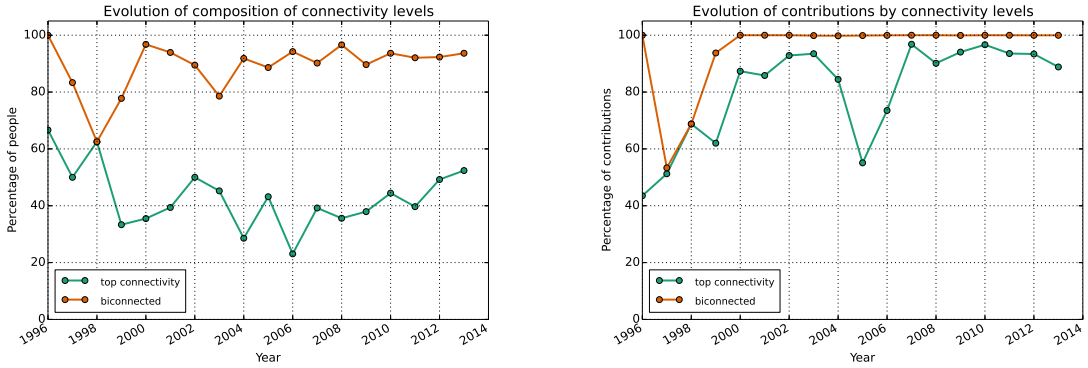
Table 5.1: Negative binomial regression

	model1	model2	model3	model4	model5
(Intercept)	-0.6471^{*}	-0.5445^{**}	-0.8369^{***}	-0.6897^{***}	-0.5353^{**}
	(0.2615)	(0.1994)	(0.2004)	(0.1827)	(0.1830)
psizes	0.3088^{***}	0.2288^{***}	0.2403^{***}	0.1695^{***}	0.1589^{***}
	(0.0207)	(0.0156)	(0.0156)	(0.0144)	(0.0144)
bugs	-0.0070^{***}	0.0016	0.0024	0.0017	0.0013
	(0.0019)	(0.0014)	(0.0014)	(0.0012)	(0.0012)
deps	-0.0203^{**}	-0.0163^{**}	-0.0174^{**}	-0.0101	-0.0090
	(0.0077)	(0.0058)	(0.0057)	(0.0053)	(0.0052)
tenure	0.1047^{***}	0.0714^{***}	0.0767^{***}	0.0855^{***}	0.0820^{***}
	(0.0079)	(0.0065)	(0.0065)	(0.0058)	(0.0059)
degree		0.0358^{***}	0.0358^{***}	0.0227^{***}	0.0223^{***}
		(0.0005)	(0.0005)	(0.0006)	(0.0006)
closeness		-0.3036	-0.1414	1.1088^{***}	1.1381^{***}
		(0.1786)	(0.1801)	(0.1727)	(0.1722)
clus_sq			0.4212^{***}		-0.3563^{***}
			(0.0772)		(0.0840)
factor(knum): 2/1				1.1343^{***}	1.2795^{***}
				(0.0545)	(0.0656)
factor(knum): 3/1				1.7000^{***}	1.7869^{***}
				(0.0777)	(0.0795)
factor(knum): 4/1				1.1306^{***}	1.2546^{***}
				(0.1527)	(0.1548)
factor(knum): 5/1				1.1962^{***}	1.3160^{***}
				(0.2268)	(0.2275)
factor(knum): 6/1				1.1454^{***}	1.2641^{***}
				(0.2972)	(0.2970)
Log-likelihood	-8234.1633	-7540.3877	-7527.7353	-7295.8007	-7287.6729
N	1750	1750	1750	1750	1750
AIC	16480.3266	15096.7754	15073.4706	14617.6015	14603.3459

The fact that the quantification of contributions in the Debian project is a discrete variable —number of package uploads to the Debian repository— restricts the options of regression modeling available. The over dispersed negative binomial regression is clear in that the connectivity level in the collaboration network has a positive and significant impact on the level of contribution of each developer. However it does not allow to take into account unobserved individual differences among developers that might explain their level of contribution. As discussed above, the data from CPython project allows us to measure contributions as lines of source code added by each developer. This variable can be safely considered continuous and therefore we can model it as a panel regression with individual fixed effects.

Let's first take a look at the descriptive data on percentage of contributions by the top connectivity level (the developers that are in the k-component with the highest k) in the CPython project. Figure 5.2a displays the evolution of percentage of developers that are at the top connectivity level throughout the history of Python project. The green line shows the percent-

age of developers that are included in the giant bicomponent, and the blue line represents the percentage of developers in the top connectivity level. We can see that around 40% of the developers that have contributed some code are in the top level of the connectivity hierarchy, and this percentage is quite stable trough time. Note that the actual k value of the top level varies in time, depending on how the patterns of relations among developers and source code files have evolved each concrete year. The node connectivity of the k-component in the top of the connectivity hierarchy is almost all years 10 or 11, with a minimum of 6 in 2005.



(a) Evolution of composition of connectivity levels (b) Evolution of contributions by connectivity levels

Figure 5.2: Evolution of the percentage of nodes in each connectivity level (left) and evolution of the percentage of contributions by developers in each connectivity levels (right) in the Python project.

Figure 5.2b shows the evolution of the contributions of the developers by connectivity level, measured in terms of lines of source code added to the project. The green line represents the percentage of contributions performed by developers in the giant biconnected component, and the blue line represents the percentage of contributions coming from developers in the top level of the connectivity hierarchy. Note that k-components are nested inside each other, like Russian dolls, thus the contributions of developers in the giant biconnected components also include the contributions of the developers in the top connectivity level. As we can see, developers in the giant biconnected component are the authors of almost all contributions, but they are also between 80% and 97% of all developers.

However developers in the top connectivity level are only around 40% of all developers on the project, but they are the authors, most years, of around 90% of the source code contributions. Some years their percentage of contributions is lower (around 60%) but this is mostly before 2001, when the community was much smaller than in the following years. Therefore only a small fraction of the developers are responsible of the lion's share of the work. Note that the actual k value of the top level varies in time, depending on how the patterns of relations among developers and source code files have evolved each concrete year. The node connectivity of the k-component in the top of the connectivity hierarchy is almost all years 10 or 11, with a minimum of 6 in 2005.

For modeling individual contributions to the CPython project, we used a panel regression with individual fixed effects. This design allows us to account for unobserved variability among the individual developers, such as cultural background or coding expertise, and disentangle if the position of a developer in the connectivity hierarchy has an effect in her level of contribution to the project. As we can see in the table, being in the top connectivity level has a positive and significant impact in the level of contribution of each developer. Note also that considering the k-number of the developer (ie, the level k of the highest k-component in which the developer is embedded) adds explanation power on the model and suggest that the impact of the connectivity hierarchy on the productivity of developers operates at all connectivity levels, not only at the top. The model also includes control variables for the centrality of each developer in the collaboration network (Degree centrality and Betweenness), the number of direct collaborators of each developer (collaborators), the tenure of each developer (measured as the number of years since the first contribution) and the value of square clustering which is a measure of local cohesion.

Table 5.2: Contributions Panel Regression Results

	Dependent variable:			
	Lines of Source Code			
	(1)	(2)	(3)	(4)
Degree Centrality	4.645^{***}	4.642^{***}	4.150^{**}	3.084^{**}
	(1.348)	(1.344)	(1.323)	(1.197)
Collaborators	0.085^{***}	0.085^{***}	0.072^{***}	0.035^{***}
	(0.006)	(0.006)	(0.006)	(0.007)
Tenure (years)	-0.269^{***}	-0.268^{***}	-0.212^{***}	-0.140^{***}
	(0.028)	(0.028)	(0.027)	(0.031)
Betweenness	1.747^{*}	1.752^{*}	1.355	2.249^{*}
	(0.776)	(0.781)	(0.755)	(0.927)
Square clustering		0.020	0.268	0.731^{*}
		(0.354)	(0.338)	(0.324)
Top connectivity level			1.154^{***}	0.485^{**}
			(0.192)	(0.178)
κ-component number				0.356^{***}
				(0.052)
Observations	816	816	816	816
Adjusted R^2	0.394	0.393	0.424	0.474
Note:				*p<0.05; **p<0.01; ***p<0.001

This regression modeling of CPython contributions by connectivity level complements and confirms the negative binomial regression results applied to the Debian project. The k-components of the collaboration network define groups of developers that are the core of the project and are responsible for most of the contributions, both in Debian and in CPython

project. These groups are central in a topological/structural sense, but these developers are also the ones responsible for the lion's share of the contributions.

The next step is to determine if the developers on the top connectivity level are always the same people, or if there is rotation and turn over. Figure 5.3 shows a Sankey diagram where each piece of the diagram represents the number of developers in the top connectivity level for a given year; the arrows that come from the top represent the number of developers who in year $y - 1$ were not at the top connectivity level but are in the top level at year y, the arrows on the bottom represent the number of developers that are in the top connectivity level at year y but not anymore in year $y + 1$. The horizontal arrow represents the number of developers that at year y are in the top connectivity level, and continue to be there at year $y + 1$.

As we can see, there is a constant flow of developers in and out of the top connectivity level throughout the history of Python project, especially when the community is consolidated after year 2000. And this is the key element to understand the dynamics of contribution because even though a very big part of the contributions come from a small set of developers (the ones in the top connectivity level), these developers are not the same people throughout the history of the project.

We argue that this feature is what defines a dynamic hierarchy, where the positions defined by it —the connectivity subgroups in the collaboration network in this concrete analysis— have a very high rate of renewal. Thus, in a community where most of their participants do not obtain their means of subsistence from the work that they do in the community, the rapid turn over of individuals that contribute the most is a key mechanism for ensuring the long term viability of the project beyond its original founders.

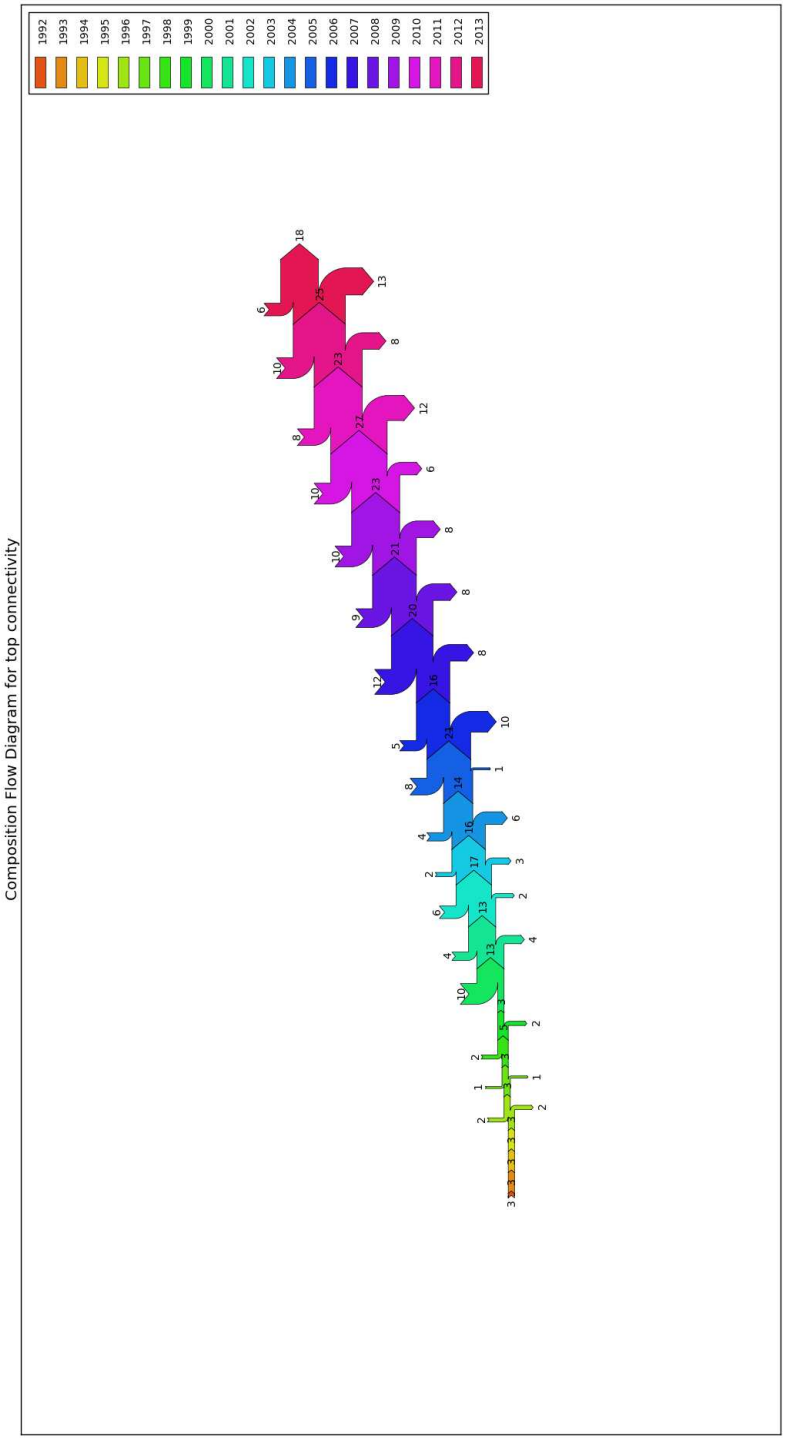


Figure 5.3: Evolution of the composition in the top connectivity level

Modeling robustness as median active live of individuals in the project

In the network literature, robustness of networks is usually measured with simulations of failures (removing nodes at random) and attacks (removing nodes decrementally starting for the ones with higher degree) (Albert et al., 2000). However this is not a good way to model the evolution of participation in a FOSS project.

We use here the survival analysis approach (Miller Jr, 2011), that according to our knowledge, is the first time that is applied to model the turn over in FOSS communities. In its origin, survival analysis, was focused on modeling lifespans of individuals and is still widely used in medicine. However, this kind of analysis can also be used to model any kind of duration. Thus we model the active life of a developer in the Python project to the period that this developer is contributing at least one line of source code. We consider a developer "dead" when she no longer contributes.

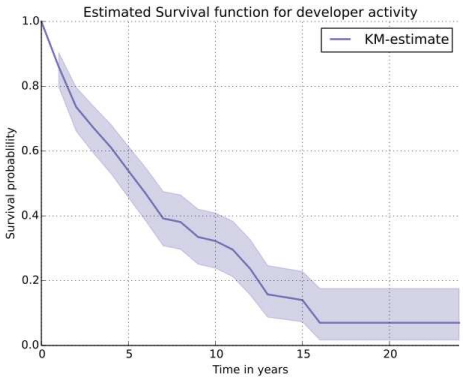
To estimate the survival function from the empirical data we used the Kaplan-Meier estimator (Kaplan and Meier, 1958) defined as:

$$\hat{S}(t) = \prod_{t_i<t} \frac{n_i - d_i}{n_i}$$

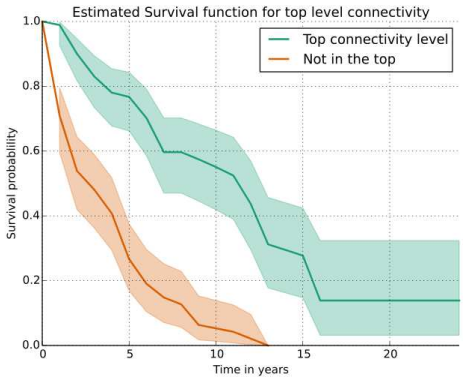
where d_i are the number of "death events" at time t and n_i is the number of subjects at risk of death at time t. If we compute the KM estimator for all developers (figure 5.4a) we can see that the median time of a developer on the community, defined as the point in time where on average half of the population has abandoned the community, is 5 years. But if we consider separately the developers in the top level of the connectivity hierarchy (figure 5.4b), their median time is 10 years; but only 3 years for the developers that are not on the top of the connectivity hierarchy.

Although it is clear that the two survival functions depicted in figure 5.4b are different, I performed the log rank test, a common statistical test in survival analysis that compares two event series' generators. The test confirms that that the two series have different generator mechanisms and are significantly different.

Finally, given that we observe an important flow of new developers towards the top levels of the connectivity hierarchy; and also having established that the contributions of the developers in these top levels is significantly higher than other developers, it is interesting to analyze the personal trajectories of developers in the project. We model the active time of developers in the Debian project using a Cox proportional hazards model with time-dependent covariates and right-censoring (Fox, 2002, appendix on survival analysis).



(a) Survival Function for all developers



(b) Survival Function for developers in the top connectivity level

Figure 5.4: Estimation of the survival function using the Kaplan-Meier estimate. The median time of a developer in the community, defined as the point in time where on average half of the population has abandoned the community, is 5 years if we consider all developers (left). But if we consider separately the developers in the top level of the connectivity hierarchy (right), their median time is 10 years; but only 3 years for the developers that are not on the top of the connectivity hierarchy.

We are interested in assessing the impact of being in the higher levels of the connectivity structure in terms of the expected active life of a developer in the project. The covariates in the model are: a) the contributions of each developer to the project, measured as number of source code lines added by each developer, b) the number of collaborators (ie neighbors in network terms) that each developer has in the collaboration network, c) the degree of each developer in the collaboration network (ie number of files of source code modified), d) the highest k of a k-component in which the developer is embedded, and c) the *Top connectivity level* dummy variable that equals to 1 if the developer is in the k-component of highest k in the collaboration network for that time period, and 0 otherwise.

In order to fit the model, we divided the data in ‘strata‘ based on the value of ‘tenure‘ covariate which reflects the time a developer has been active in the project. Each stratum is permitted to have a different baseline hazard function, while the coefficients of the remaining covariates are assumed to be constant across strata. Stratification is most natural when a covariate takes on only a few distinct values, and when the effect of the stratifying variable is not of direct interest.

Finally the estimations of the variance and standard errors of the coefficients of the covariates of interest are robust, and clustered for each developer. This is necessary because in a proportional hazards model with time-dependent covariates, each individual has more than oner row in the database. Concretely, each individual has a row for each period of one year in which he has been an active contributor to the source code of the Python project.

As we can see, the effect of being part of the top connectivity level is significant and negative, meaning that it decreases the yearly hazard of leaving the project by a factor of $e^b = e^{-1.175} = 0.31$, that is, 69%. This interpretation holds assuming that all other covariates remain constant. Notice also that the coefficient for k-number —the highest k of a k-component in

Table 5.3: Survival Analysis: Cox proportional hazards regression model

	Dependent variable:		
	Time active in the community		
	(1)	(2)	(3)
Total accepted PEPs	−0.111	−0.122	−0.107
	(0.115)	(0.118)	(0.119)
Contributions	−0.00001	−0.00001	−0.00001
	(0.00003)	(0.00003)	(0.00003)
Collaborators	0.005	−0.012	0.004
	(0.010)	(0.008)	(0.009)
Degree	−0.012	−0.015	−0.006
	(0.007)	(0.008)	(0.007)
k-component number	−0.349***		−0.286**
	(0.098)		(0.099)
Top connectivity level		−1.681**	−1.351*
		(0.623)	(0.660)
Observations	754	754	754
R^2	0.137	0.133	0.143
Max. Possible R^2	0.396	0.396	0.396
Log Likelihood	-134.570	-136.426	-132.191
Note:			*p<0.05; **p<0.01; ***p<0.001

which the developer is embedded— is also significative and negative, which means that an increment of one connectivity level decreases the yearly hazard of leaving the project by a factor of $e^b = e^{-1.769} = 0.84$, that is, 16%.

It is relevant that both measures of cohesion are significative and negative when included in the same model. We can conclude that not only being at the top connectivity level has a relevant impact on the active life of a developer in a project, but also smaller increments in cohesion of the groups in which a developer is embedded have a significant impact on their active life in the project.

5.5 Conclusion

In this paper we explored the dynamical dimension of the hierarchies that emerge on collaboration network of FOSS projects. Collaboratiuon network referes, in this context, to the patterns of relations among developers established while contributing to the project. The dynamic analysis, in this case, is not a longitudinal account of the changes in the hierarchy through time, but the analysis of the pace of renewal of individuals in the positions defined by the hierarchy. We propose that organizations —and not only FOSS projects— can be classified in

a continuum depending on the pace of renewal of the individuals that occupy key positions in the hierarchy.

We showed that the structural cohesion model (White and Harary, 2001; Moody and White, 2003) is a solid theoretical framework to define cohesive groups —k-components— in collaboration networks. The nested structure of k-components nicely captures the hierarchy in the patterns of relations that individual contributors establish when working in a FOSS project. This hierarchy, on the one hand, reflects the empirically well established fact that in FOSS projects only a small fraction of the developers account for most of the contributions. And, on the other hand, refutes the naive views of early academic accounts that characterized FOSS projects as a flat hierarchy of peers in which every individual does more or less the same.

We also showed that the position of individual developers in the connectivity hierarchy of the collaboration network impacts significantly, on the one hand, on the volume of contributions that an individual does to the project. And, on the other hand, on the median life time of developers in the project. We argue that the latter is a better way to analyze robustness of FOSS projects than the classical random and targeted attacks that has been used to asses robustness in other kinds of networks.

Our main conclusion is that the connectivity structure of FOSS collaboration networks can be characterized as a dynamic hierarchy, where the top levels of this hierarchy are filled with new individuals at a high pace. This feature is key for understanding the mechanisms and dynamics that make FOSS communities able to develop long term projects, with high individual turnover, and yet achieve high impact and coherent results. Therefore, we can conclude that cooperation in FOSS communities has a structural dimension because membership in cohesive groups that emerge from the collaboration network —the repeated patterns of relations that the direct producers establish in the production process— has an important and statistically significative impact on both the volume of individual contributions, and on the median active life of developers in a FOSS project.

Part IV

Conclusion and Future Work

6

Conclusion and Future Work

There is still a lot of work to be done.

Bibliography

Abelson, H., G. Sussman, J. Sussman, and A. Perlis (1985). *Structure and interpretation of computer programs*, Volume 2. MIT Press Cambridge, MA.

Adler, P. (2001). Market, hierarchy, and trust: The knowledge economy and the future of capitalism. *Organization Science*, 215–234.

Adler, P. and C. Heckscher (2006). *The Firm as a Collaborative Community: Reconstructing Trust in the Knowledge Economy*. Oxford University Press, Oxford, UK.

Adler, P., S. Kwon, and C. Heckscher (2008). Professional work: The emergence of collaborative community. *Organization Science 19*(2).

Ahmed, A., V. Batagelj, X. Fu, S.-H. Hong, D. Merrick, and A. Mrvar (2007). Visualisation and analysis of the internet movie database. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on*, pp. 17–24. IEEE.

Albert, R., H. Jeong, and A. Barabási (2000). Error and attack tolerance of complex networks. *Nature 406*(6794), 378–382.

Arrow, K. (1962). Economic Welfare and the Allocation of Resources for Invention. *The Rate and Direction of Inventive Activity*, 609–626.

Batagelj, V. and M. Zaveršnik (2007). Short cycle connectivity. *Discrete mathematics 307*(3), 310–318.

Batagelj, V. and M. Zaveršnik (2011). Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification 5*(2), 129–145.

Beineke, L., O. Oellermann, and R. Pippert (2002). The average connectivity of a graph. *Discrete mathematics 252*(1-3), 31–45.

Benkler, Y. (2002). Coase's penguin, or, linux and the nature of the firm. *The Yale Law Journal*.

Benkler, Y. (2006). *The Wealth of Networks*. Yale University Press.

Benkler, Y., A. Shaw, and B. M. Hill. Peer production: A modality of collective intelligence.

Blau, P. M. (1964). *Exchange and power in social life*. Transaction Publishers.

Blau, P. M. (1969). *The dynamics of bureaucracy: A study of interpersonal relations in two government agencies*. University of Chicago Press.

Blau, P. M. and W. R. Scott (1962). Formal organizations: a comparative approach.

Bolz, C., A. Cuni, M. Fijalkowski, and A. Rigo (2009). Tracing the meta-level: Pypy's tracing jit compiler. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pp. 18–25. ACM.

Borgatti, S. and M. Everett (2000). Models of core/periphery structures. *Social networks 21*(4), 375–395.

Borgatti, S., M. Everett, and P. Shirey (1990). Ls sets, lambda sets and other cohesive subsets. *Social Networks 12*(4), 337–357.

Brandes, U. and T. Erlebach (2005). *Network analysis: methodological foundations*, Volume 3418. Springer Verlag.

Castells, M. (2013). *Networks of outrage and hope: Social movements in the internet age*. John Wiley & Sons.

Coase, R. (1937). The nature of the firm. *Economica 4*(16), 386–405.

Coleman, G. (2005). Three Ethical Moments in Debian. *Social Science Research Network 805287*.

Coleman, J. (1988). Social Capital in the Creation of Human Capital. *American Journal of Sociology*, 95–120.

Csárdi, G. and T. Nepusz (2006). The igraph software package for complex network research.

Davis, G., M. Yoo, and W. Baker (2003). The small world of the American corporate elite, 1982-2001. *Strategic organization 1*(3), 301.

de Solla Price, D. (1986). *Little science, big science... and beyond*. Columbia University Press New York.

Dodds, P., D. Watts, and C. Sabel (2003). Information exchange and the robustness of organizational networks. *Proceedings of the National Academy of Sciences 100*(21), 12516.

Doreian, P. and T. Fararo (1998). *The problem of solidarity: theories and models*. Routledge.

Durkheim, E. ([1893] 2008). *The division of labor in society*. Free Press.

Ellson, J., E. Gansner, L. Koutsofios, S. North, and G. Woodhull (2002). Graphviz—open source graph drawing tools. In *Graph Drawing*, pp. 594–597. Springer.

Ferraro, F. and S. O'Mahony (2010). Managing the boundaries of an 'open' project. In *The Emergence of Organizations and Markets*. Padgett, John and Walter Powell.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports 486*(3-5), 75–174.

Fox, J. (2002). *An R and S-Plus companion to applied regression*. Sage.

Freeman, L. (1992). The sociological concept of "group": An empirical test of two models. *American Journal of Sociology*, 152–166.

Glass, R. L. (2002). *Facts and fallacies of software engineering*. Addison-Wesley Professional.

Goyal, S., M. Van Der Leij, and J. Moraga-González (2006). Economics: an emerging small world. *Journal of Political Economy 114*(2).

Grannis, R. (2009). Paths and semipaths: reconceptualizing structural cohesion in terms of directed relations. *Sociological Methodology 39*(1), 117–150.

Granovetter, M. (1985). Economic action and social structure: the problem of embeddedness. *American Journal of Sociology 91*(3), 481.

Guimerà, R., B. Uzzi, J. Spiro, and L. Amaral (2005). Team assembly mechanisms determine collaboration network structure and team performance. *Science 308*(5722), 697–702.

Gutwenger, C. and P. Mutzel (2001). A linear time implementation of spqr-trees. In *Graph Drawing*, pp. 77–90. Springer.

Hagberg, A., D. Schult, and P. Swart (2008, August). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, pp. 11–15.

Harary, F. (1969). *Graph Theory*. Addison-Wesley.

Hopcroft, J. and R. Tarjan (1974). Dividing a graph into triconnected components.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*(3), 90–95.

Jones, B., S. Wuchty, and B. Uzzi (2008). Multi-university research teams: shifting impact, geography, and stratification in science. *science 322*(5905), 1259.

Jones, E., T. Oliphant, P. Peterson, et al. (2001). SciPy: Open source scientific tools for Python.

Kamada, T. and S. Kawai (1989). An algorithm for drawing general undirected graphs. *Information processing letters 31*(1), 7–15.

Kanevsky, A. (1993). Finding all minimum-size separating vertex sets in a graph. *Networks 23*(6), 533–541.

Kaplan, E. L. and P. Meier (1958). Nonparametric estimation from incomplete observations. *Journal of the American statistical association 53*(282), 457–481.

Krackhardt, D. and J. R. Hanson (1993). Informal networks: The company behind the chart. *Harvard business review 71*(4), 104–11.

Krafft, M. (2005). *The Debian System: Concepts and Techniques*. Open Source Press.

Latapy, M., C. Magnien, and N. Vecchio (2008). Basic notions for the analysis of large two-mode networks. *Social Networks 30*(1), 31–48.

Lawler, E. (1973). Cutsets and partitions of hypergraphs. *Networks 3*(3), 275–285.

Lind, P., M. Gonzalez, and H. Herrmann (2005). Cycles and clustering in bipartite networks. *Physical Review E 72*(5), 56127.

Luccio, F. and M. Sami (1969). On the decomposition of networks in minimally interconnected subnetworks. *Circuit Theory, IEEE Transactions on 16*(2), 184–188.

Luce, R. and A. Perry (1949). A method of matrix analysis of group structure. *Psychometrika 14*(2), 95–116.

Mani, D. and J. Moody (2014). Moving beyond stylized economic network models: The hybrid world of the indian firm ownership network. *American Journal of Sociology 119*(6), pp. 1629–1669.

McFarland, D. A. (2001). Student resistance: How the formal and informal organization of classrooms facilitate everyday forms of student defiance1. *American Journal of Sociology 107*(3), 612–678.

Merton, R. K. (1979). *The sociology of science: Theoretical and empirical investigations*. University of Chicago Press.

Michels, R. (1915). *Political parties: A sociological study of the oligarchical tendencies of modern democracy*. Hearst's International Library Company.

Miller Jr, R. G. (2011). *Survival analysis*, Volume 66. John Wiley & Sons.

Mockus, A., R. Fielding, and J. Herbsleb (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology 11*(3), 309–346.

Mokken, R. (1979). Cliques, clubs and clans. *Quality & Quantity 13*(2), 161–173.

Moody, J. (2004). The structure of a social science collaboration network: Disciplinary cohesion from 1963 to 1999. *American Sociological Review 69*(2), 213–238.

Moody, J., D. McFarland, and S. Bender-deMoll (2005). Dynamic network visualization. *American Journal of Sociology 110*(4), 1206–1241.

Moody, J. and D. White (2003). Social cohesion and embeddedness: A hierarchical conception of social groups. *American Sociological Review 68*(1), 103–28.

Murdock, I. (1994). The Debian manifesto.

Newman, M. (2003). The structure and function of complex networks. *SIAM Review 45*, 167.

Nussbaum, L. and S. Zacchiroli (2010). The ultimate debian database: Consolidating bazaar metadata for quality assurance and data mining. In *7th IEEE Working Conference on Mining Software Repositories (MSR'2010)*.

O'Mahony, S. (2003). Guarding the commons: how community managed software projects protect their work. *Research Policy 32*(7), 1179–1198.

O'Mahony, S. and F. Ferraro (2007a). The emergence of governance in an open source community. *The Academy of Management Journal 50*(5), 1079–1106.

O'Mahony, S. and F. Ferraro (2007b). The Emergence of Governance in an Open Source Community. *The Academy of Management Journal (AMJ) 50*(5), 1079–1106.

Opsahl, T. (2011). Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks 34*.

Ouchi, W. (1980). Markets, bureaucracies, and clans. *Administrative science quarterly 25*(1), 129–141.

Padgett, J. F. (1990). Mobility as control: congressmen through committees. In *Social Mobility and Social Structure*, pp. 27–58. Cambridge University Press.

Palla, G., I. Derényi, I. Farkas, and T. Vicsek (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435*(7043), 814–818.

Pérez, F. and B. E. Granger (2007, May). IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng. 9*(3), 21–29.

Powell, W. (1990). Neither market nor hierarchy: Network forms of organization. *Research in Organizational Behavior 12*, 295–336.

Powell, W., D. White, K. Koput, and J. Owen-Smith (2005). Network dynamics and field evolution: The growth of interorganizational collaboration in the life sciences. *American journal of sociology 110*(4), 1132–1205.

Raymond, E. S. (1999). *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc.

Ritchie, D. M. (1993). The development of the c language. *ACM SIGPLAN Notices 28*(3), 201–208.

Robins, G. and M. Alexander (2004). Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory 10*(1), 69–94.

Roca, M. (2007). *Software libre: empresa y administración en España y Cataluña*. Edicions UOC.

Seidman, S. (1983a). Ls sets as cohesive subsets of graphs and hypergraphs. *Mathematical Social Sciences 6*(1), 87–91.

Seidman, S. (1983b). Network structure and minimum degree. *Social networks 5*(3), 269–287.

Seidman, S. and B. Foster (1978). A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology 6*(1), 139–154.

Selznick, P. (1949). *TVA and the grass roots: A study of politics and organization*, Volume 3. University of California Press.

Shaw, A. and B. M. Hill (2014). Laboratories of oligarchy? how the iron law extends to peer production. *Journal of Communication 64*(2), 215–238.

Shwed, U. and P. Bearman (2010). The temporal structure of scientific consensus formation. *American sociological review 75*(6), 817–840.

Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American philosophical society 106*(6), 467–482.

Stallman, R. M. (1985). The GNU manifesto.

Stallman, R. M. (1998). The GNU Project. *GNU project*.

Stewman, S. and S. L. Konda (1983). Careers and organizational labor markets: Demographic models of organizational behavior. *American Journal of Sociology*, 637–685.

Stiglitz, J. (1996). *Whither socialism?* The MIT Press.

Tanenbaum, A. and A. Woodhull (1996). *Operating systems: design and implementation*. Prentice Hall.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pp. 114–121. IEEE.

Torrents, J. and F. Ferraro (2015). Structural cohesion: Visualization and heuristics for fast computation. *Journal of Social Structure 16*(8).

Tönnies, F. (1974). *Community and association:(Gemeinschaft und Gesellschaft)*. Taylor & Francis.

Uzzi, B., L. Amaral, and F. Reed-Tsochas (2007). Small-world networks and management science research: a review. *European Management Review 4*(2), 77–91.

Uzzi, B. and J. Spiro (2005). Collaboration and Creativity: The Small World Problem. *American Journal of Sociology 111*(2), 447–504.

Van Rossum, G. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica.

Vedres, B. and D. Stark (2010). Structural folds: Generative disruption in overlapping groups. *American Journal of Sociology 115*(4), pp. 1150–1190.

Wasserman, S. and K. Faust (1994). *Social network analysis: Methods and applications*. Cambridge University Press.

Watts, D. (1999a). Networks, Dynamics, and the Small-World Phenomenon. *American Journal of Sociology 105*(2), 493–527.

Watts, D. (1999b). *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press.

Watts, D. and S. Strogatz (1998). Collective dynamics of 'small-world' networks. *Nature 393*(6684), 409–10.

Weber, S. (2004). *The Success of Open Source*. Harvard University Press.

White, D. and F. Harary (2001). The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, 305–359.

White, D. and M. Newman (2001). Fast approximation algorithms for finding node-independent paths in networks. *Santa Fe Institute Working Papers Series*.

White, D., J. Owen-Smith, J. Moody, and W. Powell (2004). Networks, fields and organizations: micro-dynamics, scale and cohesive embeddings. *Computational & Mathematical Organization Theory 10*(1), 95–117.

White, H. C. (1970). *Chains of opportunity: System models of mobility in organizations*. Harvard University Press Cambridge, MA.

Williamson, O. (1975). *Markets and Hierarchies, Analysis and Antitrust Implications: A Study in the Economics of Internal Organization*. Free Press.

Wuchty, S., B. Jones, and B. Uzzi (2007). The increasing dominance of teams in production of knowledge. *Science 316*(5827), 1036.

A

Small worlds and affiliation networks

In order to assert that an actual network is a small-world network we should compare it against a null model. This null model is a random network with the same number of nodes and edges, but the edges assigned uniformly at random between the nodes. A small world network should be more highly clustered than its random counterpart and it should have similar short average path length. Building in this definition of smallwordiness, we can define the small world index (Q) as the division of the CC_{ratio} by the L_{ratio} (Watts, 1999a; Davis et al., 2003; Uzzi and Spiro, 2005; Uzzi et al., 2007). If $Q > 1$ we can assert that the actual network under analysis is a small world network.

$$Q = \frac{CC_{ratio}}{L_{ratio}} \tag{A.1}$$

Where:

$$CC_{ratio} = \frac{CC_{actual}}{CC_{random}} \qquad L_{ratio} = \frac{L_{actual}}{L_{random}} \tag{A.2}$$

Affiliation networks contain two types of nodes: N actors each of which belongs to one or more groups M. Such networks are bipartite or 2-mode because they contain two types of nodes and there are no edges between nodes of same type. We can obtain an unipartite or 1-mode network —with only one type of nodes— projecting the bipartite network on the actors' side or on the groups' side. This projection assumes that the actors are connected if they belong to the same group and that groups are connected if they share some actor, respectively.

Their statistical properties differ from unipartite networks. As Uzzi et al. (2007, 83) point out, affiliation networks, on the one hand, have higher clustering than unipartite networks because each actor's membership in a team makes them a fully connected clique in the one-mode projection, therefore an important part of the clustering is not due to "the friends of an actor are friends themselves" but to team topography. On the other hand, affiliation networks tend to have shorter average path lengths as the number of overlapping members between teams increase.

Although a common practice, it is well documented in the literature that there is an important lost of information when we perform a 1-mode projection from a 2-mode network (Wasserman and Faust, 1994, 324-325). Our approach to analyze the structure of the production process of the FOSS projects is to focus on the topology and the connectivity of 2-mode networks. Robins and Alexander (2004) redefined clustering coefficient in order to analyze 2-mode networks of directors and firms. Their approach is based in the analysis of network configurations or motifs.

Figure A.1: Relevant motifs in two mode network in order to calculate CC_4 (Robins and Alexander, 2004, 78)

Figure A.1 depicts the two relevant motifs or configurations in order to compute the bipartite clustering coefficient (CC_4). Three-paths (L_3) are composed by four nodes —two of each type in the 2-mode network— linked by three edges. Robins and Alexander argue that the number of L_3 in 2-mode network is information lost in the 1-mode projection, they stress their importance: "three-paths are important to connectivity, potentially providing short geodesics between directors and companies of which they are *not* members. Long paths across the network of course must comprise several of these short three-paths, so we argue that the three-paths are precursors of global connectivity" (Robins and Alexander, 2004, 77-78).

Squares (C_4) are composed by four nodes —two of each type— linked by four edges. Squares are the simplest form of cycle in 2-mode networks and provide redundancy: when we perform the 1-mode projection, those four edges are represented by only one edge between the two nodes of the same type[1]. Robins and Alexander (2004, 79) propose compute the bipartite clustering coefficient as depicted in equation A.3.

$$CC_4 = \frac{4 \times C_4}{L_3} \tag{A.3}$$

Robins and Alexander (2004, 79) argue that "high bipartite clustering indicates localized closeness and redundancy, just as is the case with triangles in 1-mode networks. [..] If the bipartite clustering coefficient is high, then many L_3 patterns are redundant. They do not provide new paths of connectivity across the bipartite graph. So for two bipartite graphs of similar size, the graph with the higher bipartite clustering ratio will show lower levels of connectivity".

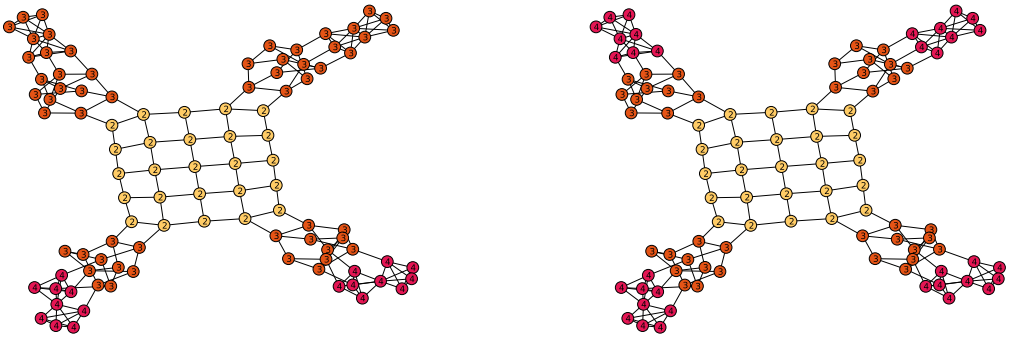
[1] If the projection is weighted the value of the edge will be 2, acknowledging that the two nodes are linked to two groups in the 2-mode network.

B

Cohesive Subgroups: Illustration, Implementation and Accuracy

B.1 Illustration of the heuristics

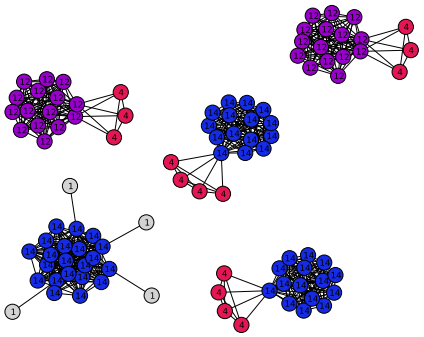
In order to illustrate how the proposed heuristics works, we will use a convenient synthetic network with 99 nodes and 200 edges where $\kappa \neq \delta$. This network is based on a two dimensional grid of 5 by 5 nodes. In each corner of the grid we attach a Petersen graph (P), linked by two edges to the grid. Thus the only four nodes of the grid with degree 2 are linked to a Petersen graph. All nodes of the grid are therefore part of a 3-core. Each P is linked to two complete graphs with 5 nodes (K_5); in two cases those two K_5 overlap in only one node and in the other two cases, they overlap in two nodes. The Petersen graph is linked by three edges to one of the K_5, thus making one of each K_5 part of a tricomponent along with P. In the case of the two K_5 that overlap only on one node, the outer K_5 has also one edge linking one of its nodes with one node of P nodes, in order to make the whole graph biconnected (see figure B.1). Petersen graphs have node connectivity 3 and complete graphs with 5 nodes have node connectivity 4. Notice that the whole example graph is biconnected and a 3-core, but it has three levels of node connectivity: 2 for the grid, 3 for the Petersen graphs (P) and 4 for the complete graphs of 5 nodes (K_5).



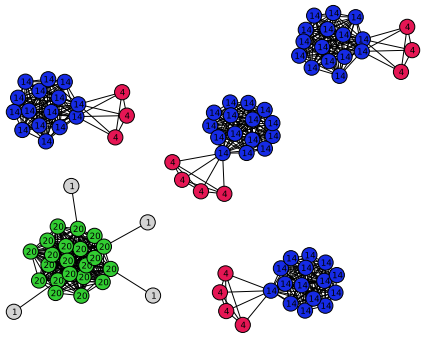
(a) Nodes colored by component number according to our algorithm. Note the error when two K_5 overlap in two nodes

(b) Nodes colored by component number according to Moody & White algorithm.

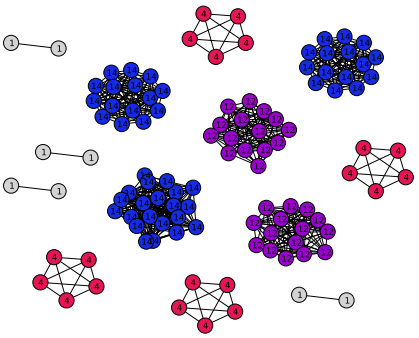
Figure B.1: Synthetic graph composed of a two dimensional grid of 25 nodes, four Petersen graphs (P) with ten nodes each (with $\kappa = 3$) linked by two edges to the grid, and eight complete graphs K_5 (with $\kappa = 4$) linked by three edges to each Petersen graph. In two cases K_5 overlap in 1 node and in the other two cases they overlap in 2 nodes. The whole graph is biconnected and also a tricore. Notice that our algorithm fails to classify the two K_5 that overlap in two nodes as 4-components. See text and figure figure B.3 for details.



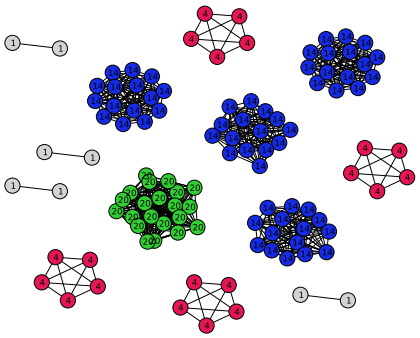
(a) Auxiliary graph H for $k = 3$ computed using White & Newman's approximation algorithm for local node connectivity.



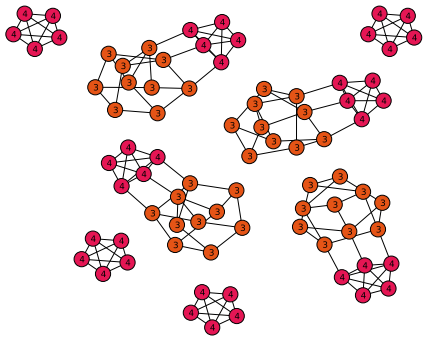
(b) Auxiliary graph H for $k = 3$ computed using flow-based connectivity algorithm for local node connectivity.



(c) All subgraphs $H_{candidate}$ from H_3 computed using White & Newman's approximation algorithm for local node connectivity.

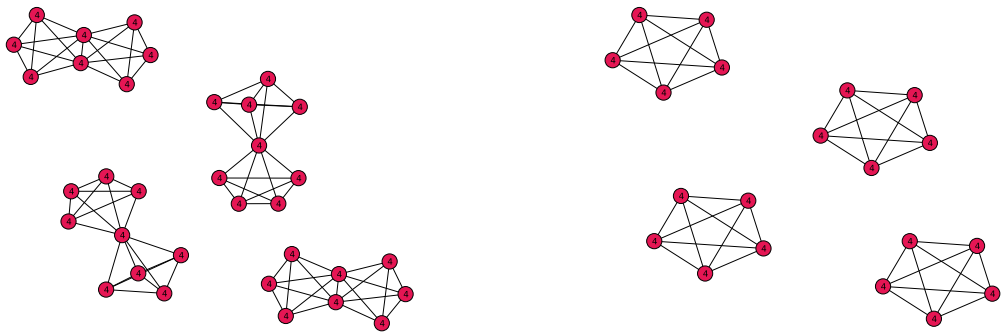


(d) All subgraphs $H_{candidate}$ from H_3 computed using flow-based connectivity algorithm for local node connectivity.



(e) Detected tri-components using the heuristics with the relaxation criteria of density ≥ 0.95 in $H_{candidate}$.

Figure B.2: Auxiliary graph H_3 for $k = 3$. Note that when using White and Newman's approximation algorithm for local node connectivity (subfigure a), some node independent paths are not detected: the P subgraphs linked to the two K_5 that overlap in two nodes should have core number 14 (blue) as in subfigure b, but they have core number 12. Thus to correctly detect all tricomponents we have to set a relaxation criteria for $H_{candidate}$, in this example setting density at 0.95 or allowing a variation of 2 in the degree of all nodes of $H_{candidate}$, allows the algorithm to correctly detect all tricomponents.



(a) Auxiliary graph H for $k = 4$ computed using White and Newman's approximation.

(b) Detected 4-components using our heuristics. Note that there should be four more K_5, the ones that overlap in two nodes are not detected as 4-components. See text for an explanation.

Figure B.3: Auxiliary graph H_4 for $k = 4$. In this case both White and Newman's approximation algorithm, and the exact flow-based algorithm for local node connectivity yield equal results. Note that there should be four more K_5 in subfigure b, the ones that overlap in two nodes are not detected as 4-components. This is because, as can be seen in subfigure a, the nodes in these $H_{candidate}$ subgraphs have all the same core number, but their density is 0.67 and the difference in degree is 3. Thus, in order to detect them we would have to relax the clique criteria for $H_{candidate}$ too much, and even then we would classify both K_5 as a single 4-component, which is obviously wrong.

As discussed above, a k-core is a maximal subgraph that contains nodes of degree k or more. The core number of a node is the largest value k of a k-core containing that node. On the other hand, a k-component is a maximal subgraph that cannot be disconnected by removing less than k nodes. The component number of a node is the largest value k of a k-component containing that node.

The graph of figure B.1 is a biconnected 3-core, which means that it is a graph with minimum degree $= 3$ that cannot be disconnected by removing less than 2 nodes. Our algorithm starts by considering the whole graph the step 2, but in k-core subgraphs with more than one bicomponent, the following steps are performed for each bicomponent of the k-core. We will only compute up until $k = 4$ because the largest core number of a node in G is 4.

For $k = 3$ we create an auxiliary graph with all biconnected nodes with core number ≥ 3 (see figure B.2). In this case all nodes have a core number greater than or equal to 3. Thus the auxiliary graph H for $k = 3$ contains all 99 nodes. We then link two nodes in H_3 if we can find k or more node independent paths between them. As we can see, the result are five connected components, four of which correspond to each Petersen graph plus the two K_5, while the last one corresponds to the nodes that form the grid. The later has 4 nodes that are linked by 3 node independent paths to only one node, these four nodes are the four corner nodes of the grid.

Notice that when using White and Newman's approximation algorithm for local node connectivity (subfigure B.2a), some node independent paths that actually exist are not detected:

the P subgraphs linked to the two K_5 that overlap in two nodes should have a core number of 14 (blue) because there are 3 node independent paths linking each pair of different nodes in the subgraph formed by the P and the K_5 to which it is linked through three edges, as in subfigure B.2b, which was computed using the exact flow-based algorithm for local node connectivity. Notice also that the grid has core number 14 in B.2a but actually should be core number 20 as shown in B.2b. This illustrates the importance of computing biconnected components of H (setp 3.c) before building the subgraphs $H_{candidate}$ (step 3.d).

Figures B.2c and B.2d depict $H_{candidate}$ subgraphs, the former using White and Newman's approximation algorithm and the latter using an exact flow-based algorithm for local node connectivity. The subgraphs $H_{candidate}$ are composed by nodes that are in the same biconnected component of H and have *exactly* the same core number. Notice that in figure B.2c the P graphs linked to the two K_5 that overlap in two nodes have core number $< n - 1$ (the magenta clusters), thus they are not complete (density=0.96) and the degree of their nodes is not homogeneus: two nodes have degree 12, four have degree 13, and nine have degree 14. Therefore, if we enforce the clique critera for $H_{candidate}$ we would not detect all tricomponents because, following the algorithm, we would have to start removing nodes with the lowest degree and check if at some point we find a complete subgraph. In order to correctly detect all tricomponents in this illustrative example, we have to first establish a relaxation for the clique criteria for $H_{candidate}$. In this case, setting density at 0.95 or allowing a variation of 2 in the degree of all nodes of $H_{candidate}$, allows the algorithm to correctly detect all tricomponents as shown in figure B.2e.

For $k = 4$, the auxiliary graph H_4 is composed of 4 connected components which correspond to the pairs of K_5 that share one node and the pairs of K_5 that share 2 nodes (see figure B.3a). In terms of biconnectivity, there are six bicomponents, with the two K_5 that overlap in two nodes as a single bicomponent. Inside these six bicomponents there are eight 4-components, but only four of them were detected (see figure B.3b). This is because when we build the $H_{candidate}$ subgraphs with all nodes in each biconnected component of H_4 that have exactly the same core number, in the case of the two K_5 that overlap in two nodes, all their nodes have the same core number (4), but their density is 0.67 and the difference in degree is 3. Thus, in order to detect them we would have to relax the clique criteria for $H_{candidate}$ too much, and even then, we would classify both K_5 overlaping in two nodes as a single 4-component, which is obviously wrong because they have node connectivity 2.

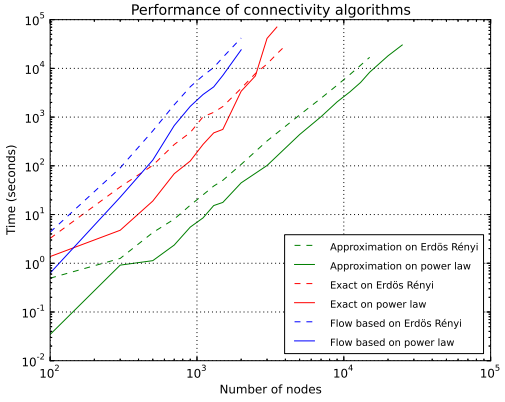
Note that this kind of false negative only happens when two k-components of the same level of connectivity and the same order overlap. If instead of two K_5 they were k-components with different order but the same connectivity, our algorithm would be able to separate them because they would have a different core number and thus they would be part of a different $H_{candidate}$ subgraph.

B.2 Performance analysis

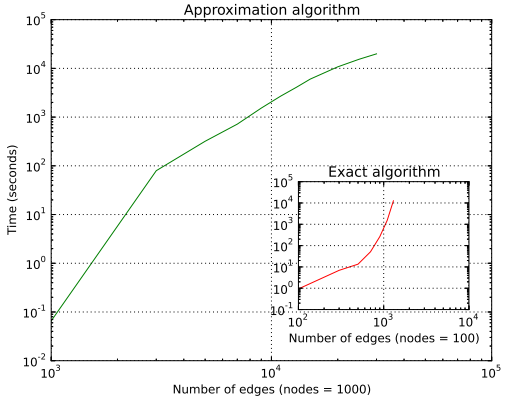
The heuristics presented here are implemented on top of NetworkX (Hagberg et al., 2008), a library for the analysis of complex networks, using the Python programming language (Van Rossum, 1995). We have chosen Python because it is a language with high readabil-

ity and flexibility that allows you to easily apply the well know principle of writing software for people to read and, only incidentally, for machines to execute (Abelson et al., 1985). To ensure reproducibility and accessibility we have used only free software to build and run all analyses presented in this paper.

The implementation of the heuristics presented here is not trivial; a careful implementation is needed to ensure that it has a reasonable memory footprint and that it runs in a reasonable time. Appendix C contains a detailed discussion of the implementation details and appendix D contains the python code of a simplified implementation for illustrative purposes.



(a) Performance of connectivity algorithms when adding nodes maintaining constant the average degree (Erdös-Rènyi) or the exponent of the power law governing the degree distribution ($\alpha = 2$). Logarithmic scale.



(b) Performance of the heuristics when adding edges and maintaining nodes constant (1000 nodes). Inset: performance of the exact algorithm with one order of magnitude fewer nodes (100 nodes). Both in logarithmic scale.

Figure B.4: Loglog plots for comparing between the heuristics and the exact algorithm to compute k-component structure. In this comparison, the heuristics do not compute the average node connectivity, only plain node connectivity, which is what is calculated by the exact algorithm. We have also implemented the exact algorithm in order to be able to compare both algorithms using the same language and infrastructure. All figures presented here were obtained running PyPy (Bolz et al., 2009). Using the heuristics proposed in this paper, we are able to handle networks almost one order of magnitude bigger than with the exact algorithm.

Figure B.4 presents the performance of the heuristics (green) compared with two variants of the exact algorithm: the Moody & White algorithm based on k-cutsets (red) and our algorithm using exact flow-based node connectivity for building the auxiliary graph. The tests were performed, on the one hand, on random graphs with fixed average degree (Erdös-Renyi model) and fixed power law exponent (Power law model) of several different orders. And, on the other hand, for graphs with a fixed number of nodes (1000 for the heuristics and 100 for the exact) where we increase the number of edges. Random networks built using the Erdös-Renyi model have a flat hierarchical structure because edges are evenly distributed across all nodes of the network. The Erdös-Renyi graphs used in this benchmark have a big tricomponent and no higher connectivity levels. Random networks built using a power law based degree distribution have a steep hierarchical structure, the networks used in the benchmark

have hierarchy levels of up to 20. Both the heuristics and the exact algorithms perform better in sparse networks with a steep hierarchical structure.

As we can see in figure B.4 the heuristics runs in polynomial time. It is fast enough to be practically applicable to networks with a few tens of thousands of nodes and edges. This is one order of magnitude better than the exact algorithm proposed by Moody and White (2003), and also an order of magnitude faster than using flow-based algorithms for building the auxiliary graph. Notice that the k-cutset based algorithm proposed by Moody & White (or at least our implementation) is faster than the exact flow-based local node connectivity variant of our algorithm.

The implementation that we provide in this paper only considers the exact solution for biconnected components. The heuristics presented here uses biconnectivity, but can be improved by using a triconnectivity algorithm. It would be: a) faster because there is a linear algorithm to compute triconnected components (Hopcroft and Tarjan, 1974; Gutwenger and Mutzel, 2001); and, b) more accurate, because we compute the exact solution up to $k = 3$. But, as far as we know, there is no publicly available implementation of triconnected components. An optimal implementation of the heuristics presented here would have to incorporate the triconnectivity algorithm to improve its accuracy and to allow it to run in reasonable time on somewhat larger networks.

B.3 Implementation details

The implementation of the heuristics proposed here was done by the first author listed on the NetworkX python library (Hagberg et al., 2008), a Python package for the study of the structure and dynamics of complex networks. Other parts of the powerful Python (Van Rossum, 1995) scientific computing stack (Jones et al., 2001; Pérez and Granger, 2007; Hunter, 2007) were also essential. The main requirement was that the whole software stack must be free software in order to avoid the black box effect of software solutions that do not release their source code. We belive that this is a necessary condition for ensuring the reproducibility of scientific research. Appendix B contains python code for the main part of the algorithm.

The implementation of the heuristics is not trivial. There are a few questions that need to be addressed in order to obtain a performance —both in terms of computation time and memory consumption— that will allow for these heuristics to be applied to large networks. The authors are in-debted to Aric Hagberg and Dan Schult (developers of the NetworkX package) for their help in this implementation.

The second step of the heuristics (compute the biconnected components of the input graph and use them as a baseline for k-components with $k > 2$) is faster than using the logic of the heuristics for $k = 2$. Biconnected components computation runs in linear time in respect to the number of nodes and edges (Tarjan, 1972). Besides in large networks, bicomponents are formed by an important part of the nodes of the network. Thus if we use the approximation logic to compute them, the memory footprint for large networks is too large to be practical. The implementation provided with this paper only computes the exact solution for bicomponents but there is also a linear algorithm to compute triconnected components (Hopcroft and Tarjan, 1974; Gutwenger and Mutzel, 2001). The heuristics would be even faster if we applied the approach used for bicomponents to that of tricomponents. But the implementation of triconnectivity is quite challenging and, to our knowledge, there is no implementation of triconnected components in free network analysis software packages.

The auxiliary graph H is usually very dense in real world networks because a large part of nodes that are in a biconnected part of a k-core are actually part of a k-component. The memory footprint of creating this dense auxiliary graph prevents a naive implementation of the heuristics in order to be practical for large networks. Our solution for this problem is to use a complement graph data structure that only stores information on the edges that are *not* present in the actual auxiliary graph. When applying algorithms to this complement graph data structure, it behaves as if it were the dense version. This is the only way to have a memory footprint that will allow for the application of the heuristics presented in this paper to large networks.

B.4 Python code

This is a simplified implementation of the heuristics for illustrative purposes. A fully functional version of NetworkX package with all the code necessary to run the heuristics is available from the authors upon request.

```
# Standard python libraries
import itertools
import collections
# NetworkX library for network analysis
import networkx
# White and Newman node connectivity approximation
# Code in https://networkx.lanl.gov/trac/ticket/538
from connectivity_approx import vertex_connectivity_approx
# AntiGraph data structure
# code in https://networkx.lanl.gov/trac/ticket/608
import antigraph

def k_components(G, average=True, exact=False, min_density=0.95):
    def _update_results(k, avg_k, components):
        # Auxiliary function to update results data structures
        # Code not shown
    if exact:  # Use flow based exact algorithm
        node_connectivity = nx.local_node_connectivity
    else: # Use White and Newman (2001) approximation algoritm
        node_connectivity = local_node_connectivity
    ## Data structures to return results
    # Dictionary with connectivity level (k) as keys and a list of
    # sets of nodes that form a k-component as values
    k_components = collections.defaultdict(list)
    # Dictionary with nodes as keys and maximum k of the deepest
    # k-component in which they are embedded
    k_number = dict( ((n,(0,0)) for n in G) )
    # dict to store node independent paths
    nip = {}
    ##################
    # Exact solution for k = 1
    components = networkx.connected_components(G)
    _update_results(1, 1, components)
    # Bicomponents as a base to check for higher order k-components
    bicomponents = networkx.biconnected_components(G)
    _update_results(2, 2, bicomponents)
    # There is no k-component of k > maximum core number
    # \kappa(G) <= \lambda(G) <= \delta(G)
    g_cnum = core_number(G)
    max_core = max(g_cnum.values())
    for k in range(3, max_core + 1):
        C = k_core(G, k, core_number=g_cnum)
        for nodes in biconnected_components(C):
            # Build a subgraph SG induced by the nodes that are part of
            # each biconnected component of the k-core subgraph C.
            if len(nodes) < k:
                continue
```

```
48              SG = G.subgraph(nodes)
49              # Build auxiliary graph
50              H = AntiGraph()
51              H.add_nodes_from(SG.nodes_iter())
52              for u,v in combinations(SG, 2):
53                  K = node_connectivity(SG, u, v)
54                  nip[(u,v)] = K
55                  if k > K:
56                      H.add_edge(u,v)
57              for h_nodes in biconnected_components(H):
58                  if len(h_nodes) <= k:
59                      continue
60                  HS = H.subgraph(h_nodes)
61                  h_cnum = core_number(HS)
62                  first = True
63                  for c_value in sorted(set(h_cnum.values()),reverse=True):
64                      cands = set(n for n, cnum in h_cnum.items() if cnum == c_value)
65                      # Skip checking for overlap for the highest core value
66                      if first:
67                          overlap = False
68                          first = False
69                      else:
70                          overlap = set.intersection(*[
71                                      set(x for x in HS[n] if x not in cands)
72                                      for n in cands])
73                      if overlap and len(overlap) < k:
74                          Hc = HS.subgraph(cands | overlap)
75                      else:
76                          Hc = HS.subgraph(cands)
77                      if len(Hc) <= k:
78                          continue
79                      hc_core = core_number(Hc)
80                      if _same(hc_core) and density(Hc) == 1.0:
81                          Gc = k_core(SG.subgraph(Hc), k)
82                      else:
83                          while Hc:
84                              Gc = k_core(SG.subgraph(Hc), k)
85                              Hc = HS.subgraph(Gc)
86                              if not Hc:
87                                  continue
88                              hc_core = core_number(Hc)
89                              if _same(hc_core) and density(Hc) >= min_density:
90                                  break
91                              hc_deg = Hc.degree()
92                              min_deg = min(hc_deg.values())
93                              remove = [n for n, d in hc_deg.items() if d == min_deg]
94                              Hc.remove_nodes_from(remove)
95                      if not Hc or len(Gc) <= k:
96                          continue
97                      for k_component in biconnected_components(Gc):
98                          if len(k_component) <= k:
99                              continue
```

```
100                         Gk = k_core(SG.subgraph(k_component), k)
101                         num = 0.0
102                         den = 0.0
103                         for u,v in combinations(Gk, 2):
104                             den += 1
105                             num += (nip[(u,v)] if (u,v) in nip
106                                     else nip[(v,u)])
107                         _update_results(k, [Gk.nodes()], (num/den))
108     return k_components, k_number
```

B.5 Accuracy and limitations of the heuristics

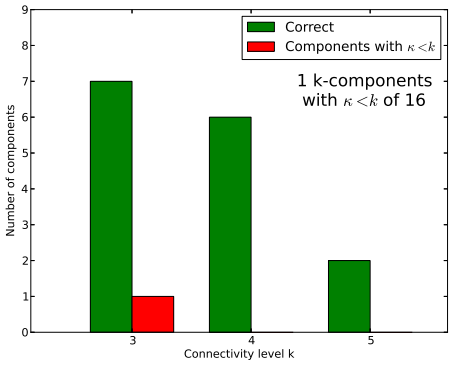
Figure B.5 shows the accuracy of connectivity structure detected by the heuristics for all empirical networks. In the subfigures, green bars are k-components with node connectivity $\geq k$ and red bars represent k-components with node connectivity $< k$. Note that, once we have an approximate structure of k-components, we can check —in a reasonable time frame— if the resulting k-components actually have node connectivity k using flow based connectivity algorithms (Brandes and Erlebach, 2005, chapter 7). For the candidate k-components that turned out to have node connectivity lower than k, we used the exact algorithm proposed by Moody and White (2003) to find out the order and size of the actual k-components inside the candidate k-component detected using our heuristics.

The output of our heuristics is an approximation to k-components based on computing extra-cohesive blocks for each biconnected component of all core levels of the network. Recall that in k-components all k node independent paths go through nodes that belong to the k-component, but in extra-cohesive blocks some of the node independent paths may go through external nodes. Thus, there is no guarantee that the extra-cohesive blocks, even those that also form a k-core subgraph in G, have node connectivity $\kappa = k$. This is a source of false positives for the approximation of the k-component structure of a network. However, the results shown in figure B.5 suggest that the heuristics yield a good approximation for the actual —k-component based— cohesion structure of empirical networks.

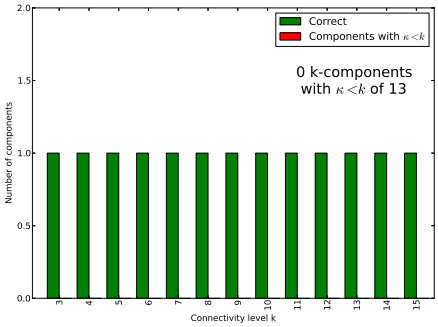
If we consider all components of all sizes, as in figure B.5, only a few of the extra-cohesive blocks detected by the heuristics have node connectivity of less than k, ranging from 6.5% (a single component) in the case of Debian to 1.2% of the components in the case of two-mode Nuclear Theory network. However, the extra-cohesive blocks that do not have the sufficient connectivity to be considered a k-component are, in the empirical networks analyzed, big components of levels {3,4}. This is because, in such big- and low-level components, a few node independent paths going through nodes that are part of the biconnected component of a k-core but not part of the k-component can yield false positives by including nodes that shouldn't be part of the k-component.

However, these false positives are actually part of an extra-cohesive block, which maintains most of those properties —in terms of robustness, hierarchy and overlap— which make k-component such a good measure of structural cohesion. This relaxed definition of connectivity might be sufficient in many cases; for instance, if we are interested in comparing the structural cohesion of a large network with a suitable null model, we may not need the exact k-component structure because we can meaningfully compare the relaxed connectivity structure of the actual network with its random counterparts. However, imagine we are interested in the exact k-component structure of a particular network because, say, we want to statistically analyze the impact of the connectivity level with the performance of different actors in a network. In this case, we would need to apply some cutting procedure on the extra-cohesive blocks that actually have a node connectivity of less than k.

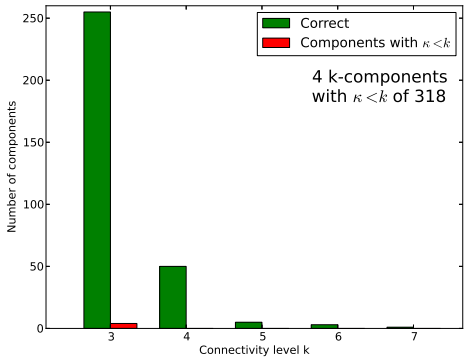
It is more difficult to assess the impact of false negatives —that is, nodes that should be part of a k-component but are excluded— because computing exact k-components for big networks is not practical, and thus we cannot compare. False negatives are derived from the underestimation of local node connectivity of the White and Newman (2001) algorithm, which



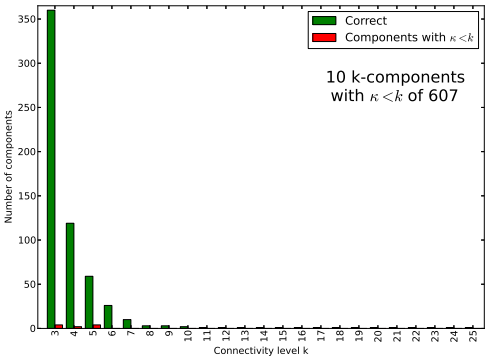
(a) Bipartite network formed by developers and packages over 2 years of collaboration (from 2007 to 2009) on the release codenamed Lenny of the Debian operating system



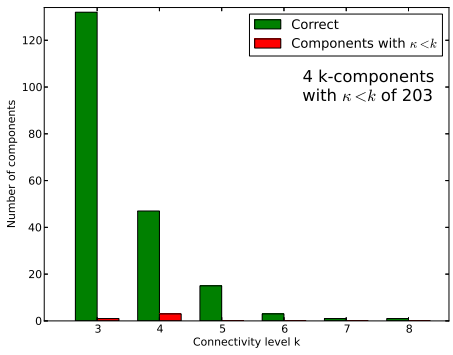
(b) Unipartite network formed by developers over 2 years of collaboration (from 2007 to 2009) on the release codenamed Lenny of the Debian operating system



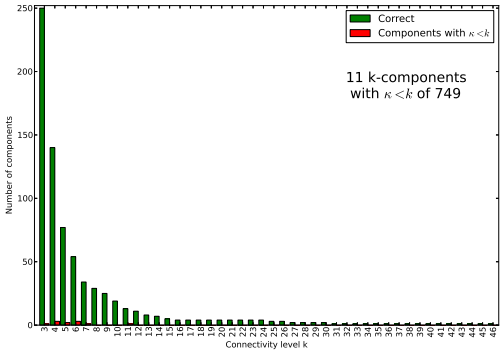
(c) Bipartite network formed by scientists and preprints during a five-year period (2006-2010) in the high energy physics (theory) section of arXiv.org



(d) Unipartite network formed by scientists during a five-year period (2006-2010) in the high energy physics (theory) section of arXiv.org



(e) Bipartite network formed by scientists and preprints during a five-year period (2006-2010) in the nuclear physics (theory) section of arXiv.org



(f) Unipartite network formed by scientists during a five-year period (2006-2010) in the nuclear theory section of arXiv.org

Figure B.5: Accuracy barplots. Green bars are k-components with node connectivity $\geq k$ and red bars represent k-components with node connectivity $< k$.

provides a strict lower bound for the local node connectivity. Thus, by using it we can miss an edge in the auxiliary graph H that should be there. Therefore, a node belonging to a k-component could be excluded by the algorithm. Recall that in order to address this problem, we relaxed the clique criteria by setting a density threshold of 0.95 in $H_{candidate}$. Whilst this value has worked well in our analysis but careful experimentation should be performed to set this parameter in other types of networks.