Advances in Complex Systems
Vol. 17 (2014) 1450022 (26 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0219525914500222

NODE MIXING AND GROUP STRUCTURE OF COMPLEX SOFTWARE NETWORKS

LOVRO ŠUBELJ*, SLAVKO ŽITNIK†, NELI BLAGUS‡
and MARKO BAJEC§

*Faculty of Computer and Information Science,
University of Ljubljana, Tržaška Cesta 25,
SI-1001 Ljubljana, Slovenia*
*lovro.subelj@fri.uni-lj.si
†slavko.zitnik@fri.uni-lj.si
‡neli.blagus@fri.uni-lj.si
§marko.bajec@fri.uni-lj.si

Received 16 August 2013
Revised 27 June 2014
Accepted 21 August 2014
Published

Large software projects are among most sophisticated human-made systems consisting of a network of interdependent parts. Past studies of software systems from the perspective of complex networks have already led to notable discoveries with different applications. Nevertheless, our comprehension of the structure of software networks remains to be only partial. Here we investigate correlations or mixing between linked nodes and show that software networks reveal dichotomous node degree mixing similar to that recently observed in biological networks. We further show that software networks also reveal characteristic clustering profiles and mixing. Hence, node mixing in software networks significantly differs from that in, e.g., the Internet or social networks. We explain the observed mixing through the presence of groups of nodes with common linking pattern. More precisely, besides densely linked groups known as communities, software networks also consist of disconnected groups denoted modules, core/periphery structures and other. Moreover, groups coincide with the intrinsic properties of the underlying software projects, which promotes practical applications in software engineering.

Keywords: Software networks; node mixing; node groups; software engineering.

1. Introduction

Large software projects are one of the most sophisticated and diverse human-made systems. Still, our comprehension of their complex structure and behavior remains to be only partial [5]. On the other hand, studies on modeling software systems as networks of interdependent parts have recently led to some notable discoveries and promoted different applications [12, 46]. Complex networks possibly provide the most adequate framework for the analysis of large software systems developed according to object-oriented, structured programming and other paradigms [30, 51].

Past studies have already shown that software systems modeled as directed networks are *scale-free* [2] with a power-law in-degree distribution and, e.g., exponential out-degree distribution [52, 53]. Furthermore, networks are *small-world* [57], when represented with undirected graphs [30, 23], and reveal a hierarchical [55] and fractal structure [8, 4]. The latter can be, similarly as the properties mentioned above, related to code complexity or reusability and the quality of the underlying software projects [47, 51]. Authors have also proposed different growing models of software networks [52, 44, 27] and investigated the importance of particular nodes in the networks [23], their evolution during project execution [5], practical applications of network community and motif structure [54, 46], and other [47].

In the present paper, we first analyze the correlations or *mixing* [31, 32] between linked nodes in software networks, which has not yet been addressed properly. Despite a common belief that software networks are negatively correlated or *disassortative* by degree [32, 15] as, e.g., web graphs or the Internet [38], we show that networks are indeed strongly disassortative by in-degree, but much more positively correlated or *assortative* by out-degree, otherwise a characteristic property of different social networks [36]. Software networks thus reveal *dichotomous* degree mixing, similar to that recently detected in undirected biological networks [19].

We further show that software networks are characterized by a sickle-shaped *clustering* [57] profile also observed in [19]. This unique shape is retained in the case of *degree-corrected clustering* [43], whereas the structure of the networks differs significantly from that of the Internet or a social network. More precisely, software networks contain connected parts or regions with very low or very high degree-corrected clustering (Fig. 1), which is else observed only for either the Internet or a social network. Nevertheless, all types of networks reveal clear degree-corrected clustering assortativity that has not been reported in the literature before.

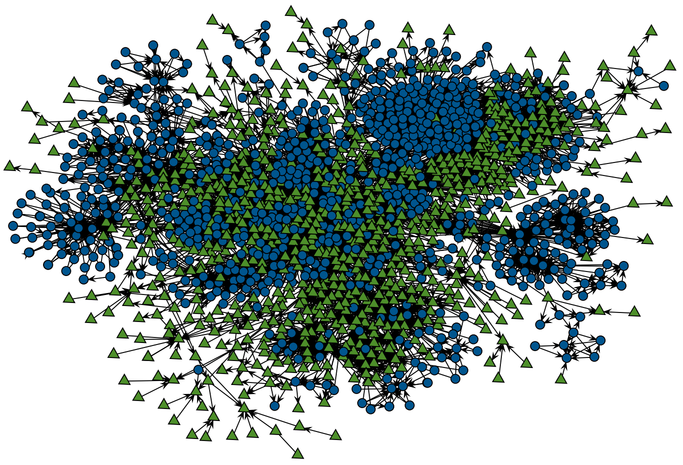


Fig. 1. Software dependency network representing the Lucene search engine. (Nodes with degree-corrected clustering [43] above or below the mean are shown as circles and triangles, respectively.)

We explain the observed degree mixing and clustering assortativity through the presence of different types of *groups* or *clusters* of nodes with common linking pattern [35]. Besides densely linked groups denoted *communities* [16], software networks also consist of groups of structurally equivalent nodes denoted *modules* [48], and different *mixtures* [49] of these, with core/periphery and hub and spokes structures as special cases. We stress that the existence of different types of groups implies high clustering assortativity, with sparse module-like groups occupying regions with very low clustering and dense community-like groups in regions with higher clustering. While the former explain the observed disassortativity by degree, the latter in fact promote the assortativity in the out-degree. Note that the conclusions are consistent with the results obtained for the Internet and a social network, where mostly module-like or community-like groups are found, respectively.

Although the main purpose of the analysis of node mixing is to relate characteristic group structure to the existing network properties, the dichotomous degree mixing in fact implies many of the common properties of real-world networks [19] (e.g., robustness). The latter, together with the observed node clustering assortativity, might be of independent interest in network model design and other.

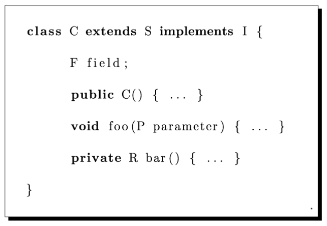
The paper does not provide a clear rationale behind the existence of different types of groups in software networks. Nevertheless, the revealed groups are found to closely coincide with some of the intrinsic properties of the underlying software projects. The paper thus also includes preliminary work and results of selected applications of network group detection in software engineering.

The rest of the paper is structured as follows. For the analysis in the paper, we adopt software dependency networks based on [46, 47], which are introduced in Sec. 2. Next, Sec. 3 contains an extensive empirical analysis and formal discussion on node degree and clustering mixing. Analysis of the characteristic groups of nodes in software networks is conducted in Sec. 4, while some practical applications of group detection in software engineering are given in Sec. 5. Section 6 concludes the paper and gives prominent directions for future work.

2. Software Dependency Networks

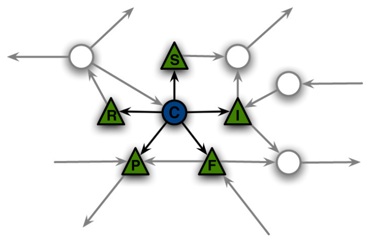
Complex software systems can be modeled with various types of networks including software architecture maps [52], class diagrams [54], inter-package [24] and class dependency networks [46], class, method and package collaboration graphs [21], software mirror [5] and subroutine call graphs [30], to name just a few. Networks mainly divide whether they are constructed from source code, byte code or program execution traces, and due to the level of software architecture represented by the nodes and the types of software relationships represented by the links.

For consistency with most past work, we consider class dependency networks [46, 47] that are suitable for modeling object-oriented software systems. Here, nodes represent software classes and links correspond to different types of dependencies among them (e.g., inheritance). More formally, let a software project consist of



```
class C extends S implements I {
        F field;
        public C() { ... }
        void foo(P parameter) { ... }
        private R bar() { ... }
}
```

(a)



(b)

Fig. 2. (a) A toy example class written in Java and (b) the corresponding class dependency network.

classes $C = \{C_1, C_2, \ldots\}$. Corresponding class dependency network is a directed graph $G(V, L)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and L is the set of links, $m = |L|$. Class C_i is represented by a node $i \in V$, while a directed link $(i, j) \in L$ corresponds to some dependency between classes C_i and C_j (Fig. 2). This can be either an *inheritance* (i.e., C_i extends class or implements interface C_j), a *composition* (i.e., C_i contains a field or variable of type C_j) or a *dependence* (i.e., C_i contains a constructor, method or function with parameter or return type C_j).

Note that class dependency networks are constructed merely from the signatures of software classes, and fields and functions therein. Thus, the networks address the inter-class structure of the software systems, whereas the intra-class dependencies are ignored [47]. However, as such information is often decided by a team of developers, prior to the actual software development, it is not influenced by the programming style of each individual developer. Moreover, such networks coincide with the flow of information and also the human comprehension of object-oriented software systems. Nevertheless, the networks still give only a partial view of the system.

According to the object-oriented programming paradigm, a class that extends a parent class also inherits all of its functionality (not considering the visibility). Hence, each class implicitly acquires the dependencies of its parent class, the parent class of its parent class, and so on. For the analysis in the paper, we thus first construct the networks based on the explicit class dependencies as described above, while we then copy also the implicit dependencies of each class from its parent classes. This provides somewhat more adequate representation of the intrinsic structure of the software system and also coincides with the developer's view. Note that the process does not significantly increase the overall number of dependencies (see below). Finally, we reduce the networks to simple directed graphs, to limit the influence of individual developers as above. Networks thus utilize merely the connectedness between the nodes, while disregarding its strength. We consider four such software dependency networks that are shown in Table 1 (see also Fig. 1). All selected networks represent well-known software projects developed in Java including physics simulation, scientific computing and network analysis libraries.

For a thorough empirical comparison in the following sections, we also consider two other real-world networks. Namely, a snapshot of communications between

Table 1. Software, Internet and social networks used in the study. (The values in brackets show the number of links corresponding to explicit class dependencies.)

Network	Description	n	m	
jbullet	JBullet 2.72 game physics simulation toolbox	166	619	(552)
colt	Colt 1.2.0 scientific and technical computing library	227	963	(709)
jung	JUNG 2.0.1 network and graph analysis framework	306	930	(713)
lucene	Lucene 4.1.0 high-performance text search engine	1657	6808	(6252)
internet	Oregon 2003 autonomous systems snapshot [26]	767	1857	—
collaboration	Network scientists collaborations [33]	1589	2742	—

Note: Software networks are reduced to largest connected components.

autonomous systems of the Internet collected by the University of Oregon in 2003 [26] and a social network of collaborations between scientists working on network theory and experiment [33] (Table 1). These are simple undirected networks. Although some directed social and technological networks would enable more straightforward comparison, such networks are commonly either much larger than software networks or do not reveal particularly clear group structure. On the other hand, we stress that the selected networks represent two fundamentally different topologies. While social networks are characterized by a dense degree assortative structure and community-like groups [31, 36], the Internet is much sparser and disassortative by degree [38]. Also, the prevalent groups of nodes are module-like, e.g., hub and spokes [25].

3. Node Mixing in Software Networks

The present section contains an extensive comparative analysis of different networks according to node degree and clustering mixing. We first review characteristics of node degree distributions in Sec. 3.1 and then show that software networks reveal dichotomous degree mixing in Sec. 3.2. Next, sickle-shaped clustering profiles of software networks are explored in Sec. 3.3, while Sec. 3.4 provides empirical evidence of node clustering assortativity in real-world networks.

3.1. *Scale-free node degree distributions*

Let k_i be the degree of node $i \in V$ and let $\langle k \rangle$ be the mean degree in the network. For directed networks, the degree is defined as the sum of in-degree and out-degree. Next, let Δ be the maximum degree, and Δ_{in} and Δ_{out} the maximum in-degree and out-degree, respectively. Last, let γ be the scale-free exponent of the power-law degree distribution $P(k) \sim k^{-\gamma}$ [2], $\gamma > 1$, and let γ_{in} and γ_{out} be the exponents corresponding to in-degree and out-degree distributions, respectively. The values of γ-s were estimated by maximum-likelihood method with goodness-of-fit tests [7].

Table 2 describes node degree sequences of different networks. The degree $\langle k \rangle$ is somewhat comparable across software networks and approximately half the size for *internet* and *collaboration* networks. Observe, however, that in the case of

Table 2. Node degree sequences of different networks. (The exponents γ-s in italics do not represent a valid fit to a power-law [7].)

Network	$\langle k \rangle$	Δ	Δ_{in}	Δ_{out}	γ	γ_{in}	γ_{out}
jbullet	7.46	62	62	22	2.80	2.26	*4.04*
colt	8.48	140	140	13	2.56	2.56	*3.91*
jung	6.08	95	92	12	2.65	2.77	*4.47*
lucene	8.22	337	333	20	2.24	2.14	*4.91*
internet	4.68	303	—	—	2.28	—	—
collaboration	3.45	34	—	—	2.85	—	—

directed software networks the values of Δ-s and γ-s are obviously governed by a much broader in-degree sequences, compared to a relatively suppressed out-degree sequences (e.g., *lucene* network). Particularly, as past work has already shown, software networks have scale-free in-degree distribution that follows a power-law with $2 < \gamma_{in} < 3$ [52] and highly truncated, e.g., log-normal [9] or exponential [53], out-degree distribution (see Table 2). Note also that the tail of the (in-)degree distribution of *lucene* software network is well modeled by the scale-free degree distribution of a sparse topology of the Internet, while, from the perspective of out-degrees, the network is somewhat more similar to a dense assortative social network (Fig. 3).

For the concerned software dependency networks, in-degree and out-degree sequences have a rather clear meaning in software engineering. The out-degree of node i corresponds to the number of classes required to implement the functionality of class C_i and is thus a measure of "external" complexity [47]. Indeed, different software quality metrics are based on the out-degrees of nodes in software networks [6, 51]. On the other hand, the in-degree of node i corresponds to the number of classes that depend on or use class C_i and is related to the level of code reusability [47].

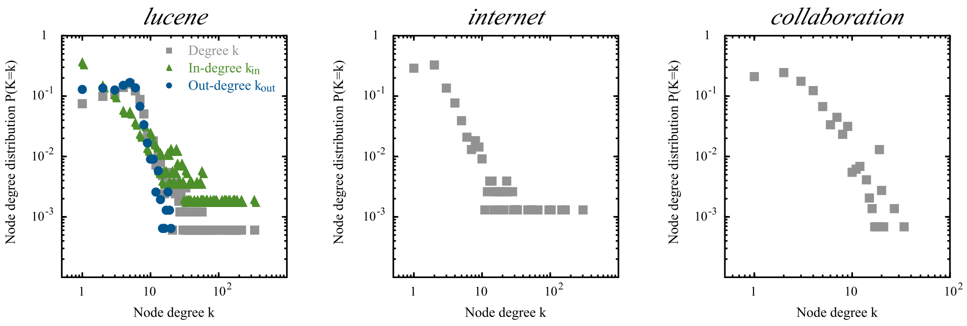


Fig. 3. Node degree distributions of larger networks (see also Table 2). Note that *lucene* software network reveals scale-free (in-)degree distribution as the Internet and a truncated, e.g., log-normal or exponential, out-degree distribution more similar to the *collaboration* network.

Highly reused classes are, obviously, well known among developers and are thus also more commonly used in the future. The latter is exactly the principle behind the preferential attachment model [2], which produces power-law in-degree distribution in software dependency networks [47]. For the case of the out-degree distribution, long scale-free tail is suppressed by constant incremental refactoring of classes within a growing software project [3] (to reduce its complexity), while such distribution also results from a certain class of software duplication mechanisms [53].

3.2. *Dichotomous node degree mixing*

The most straightforward way to analyze node degree mixing in general networks is to measure r [31, 32], which is defined as a Pearson correlation coefficient of degrees at links' ends, $r \in [-1, 1]$. Hence

$$r = \frac{1}{2\sigma_k} \sum_{(i,j)\in L} (k_i - \langle k\rangle)(k_j - \langle k\rangle), \tag{1}$$

where σ_k is the standard deviation, i.e., $\sigma_k = \sqrt{\sum_{i\in V}(k_i - \langle k\rangle)^2}$. Assortative mixing by degree shows as a positive correlation $r > 0$, while disassortative degree mixing refers to a negative correlation $r < 0$. For the case of directed networks, one can similarly define four additional coefficients $r_{(\alpha,\beta)}$ [14], $\alpha, \beta \in \{in, out\}$, where α, β correspond to the types of degrees of links' source and target nodes, respectively.

Table 3 summarizes degree mixing in different networks. As already stated before, social networks reveal strong assortative mixing [31] (e.g., *collaboration* network), whereas the Internet is degree disassortative [38]. Software networks also appear to be disassortative by degree according to r [15]. Nevertheless, this is actually a consequence of the prevailing in-degree sequences (see Sec. 3.1). The networks are indeed highly disassortative by in-degree, $r_{(in,in)} \ll 0$, though much more assortative by out-degree in most cases, $r_{(out,out)} \gg r_{(in,in)}$ (e.g., *lucene* network). Expectedly, $r_{(in,out)}$ reveals no clear mixing regime, $r_{(in,out)} \approx 0$, while $r_{(out,in)}$ is again governed by the dominant in-degrees, $r_{(out,in)} \approx r_{(in,in)}$.

Note that above coefficients provide a rather limited global view of degree mixing and can capture merely linear correlations. Figure 4 shows also neighbor connectivity plots [38] that display mean neighbor degree k_N against node degree k. Here, assortative or disassortative mixing reflects in either increasing or decreasing trend,

Table 3. Node degree mixing coefficients [15] of different networks.

Network	r	$r_{(in,in)}$	$r_{(in,out)}$	$r_{(out,in)}$	$r_{(out,out)}$
jbullet	−0.21	−0.29	−0.07	−0.26	−0.14
colt	−0.24	−0.27	−0.06	−0.25	−0.28
jung	−0.22	−0.25	−0.05	−0.24	−0.13
lucene	−0.28	−0.30	0.00	−0.29	−0.04
internet	−0.26	—	—	—	—
collaboration	0.46	—	—	—	—

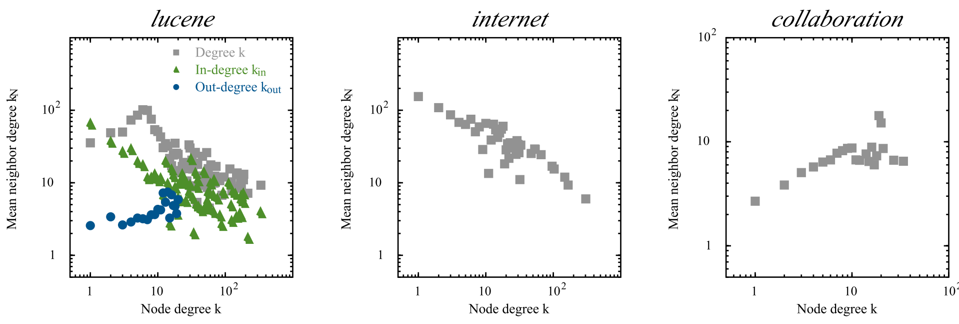


Fig. 4. Neighbor connectivity plots [38] of larger networks (see also Table 3). Note that *lucene* software network reveals dichtomous degree mixing that is disassortative by in-degree as the Internet and assortative by out-degree as social networks (e.g., *collaboration* network).

respectively. While the software network is clearly disassortative by in-degree, it is in fact slightly assortative by out-degree, as in the case of a social network. Furthermore, the degrees k show a clear two-phase or dichotomous mixing that is controlled by out-degrees for smaller k, and by in-degrees, when k increases. Although one can also observe some dichotomous behavior for *collaboration* and *internet* networks, this does not appear significant and can be due to the size of the networks. Thus, as previously claimed, software networks reveal dichotomous degree mixing and differ from other degree disassortative networks like web graphs and the Internet.

It ought to be mentioned that similar observations were recently made also in undirected biological networks [19]. Although these are disassortative by degree [29], removing a certain percentage of high degree nodes or *hubs* [18] renders the networks degree assortative. Since hubs in software networks correspond to nodes with high in-degree (see Table 2), our work generalizes that in [19] to directed networks.

Dichotomous degree mixing in software networks can be seen as a product of different programming paradigms. Recall that the out-degree of a node measures the complexity of the corresponding software class, whereas its in-degree is related to class reuse (see Sec. 3.1). Disassortativity in the in-degrees can be interpreted as low probability of hubs to link; thus, highly reused classes tend not to depend on each other. Since these commonly implement a rather different functionality, the latter is in fact a result of minimum-coupling and maximum-cohesion principle [45]. On the other hand, object-oriented software systems are commonly developed according to Lego hypothesis [3], where smaller and simpler classes are used to implement larger and more complex ones, and so on. As this results in an entire hierarchy of classes with increasing complexity across the levels of the hierarchy, a class depends only on classes with rather similar complexity, i.e., classes from the previous level. Obviously, this implies assortativity in the out-degrees in software networks.

3.3. *Sickle-shaped node clustering profiles*

Besides degree distributions and mixing considered above, real-world networks are commonly assessed due to their transitivity. For simple undirected graphs, this can be measured by node clustering coefficient c [57], $c \in [0,1]$, defined as

$$c_i = \frac{t_i}{\binom{k_i}{2}}, \tag{2}$$

where t_i is the number of links between the neighbors of node $i \in V$ and $\binom{k_i}{2}$ is the maximal number of links ($c_i = 0$ for $k_i \leq 1$). Note that the denominator in Eq. (2) introduces biases in the definition, since $\binom{k_i}{2}$ often cannot be reached due to a fixed degree sequence [43] (see below). Thus, an alternative definition of node degree-corrected clustering coefficient d [43], $d \in [0,1]$, has been proposed as

$$d_i = \frac{t_i}{\omega_i}, \tag{3}$$

where ω_i is the maximal possible number of links between the neighbors of node i with respect to their degrees ($d_i = 0$ for $k_i \leq 1$). Since $\omega \leq \binom{k}{2}$, $d \geq c$ by definition.

Table 4 shows the mean node (degree-corrected) clustering $\langle c \rangle$ and $\langle d \rangle$ in different networks. As these are small-world [57], $\langle c \rangle$ and $\langle d \rangle$ are considerably larger than the expected clustering coefficient p in a corresponding random graph [11], $p = \langle k \rangle/(n-1)$. The structure of *collaboration* network else reveals the most densely linked neighborhoods, where the majority of nodes have d equal to one (see Table 4). Exactly the opposite holds for *internet* network, where d is close to zero, $d < p$, in most cases. On the other hand, software networks are again characterized by an interplay between the dense structure of social networks and the sparse topology of the Internet. Most of the nodes have moderate values of d, $p < d < 1$, whereas nodes with either very low or high d are concentrated in certain parts of the networks (not shown).

We next consider node (degree-corrected) clustering profiles shown in Figs. 5 and 6. One can observe degree biases in the standard definition of clustering c that imply low c for hubs [see Eq. (2)], particularly apparent in degree disassortative networks (see Fig. 5). More precisely, c decreases rapidly with k, roughly following

Table 4. Node clustering coefficients of different networks.

			(% nodes)	
Network	$\langle c \rangle$	$\langle d \rangle$	$d = 1$	$d < p$
jbullet	0.43	0.50	9	20
colt	0.50	0.58	17	13
jung	0.51	0.58	19	19
lucene	0.50	0.55	11	13
internet	0.29	0.32	21	55
collaboration	0.64	0.69	61	28

Note: Networks are reduced to simple undirected graphs.

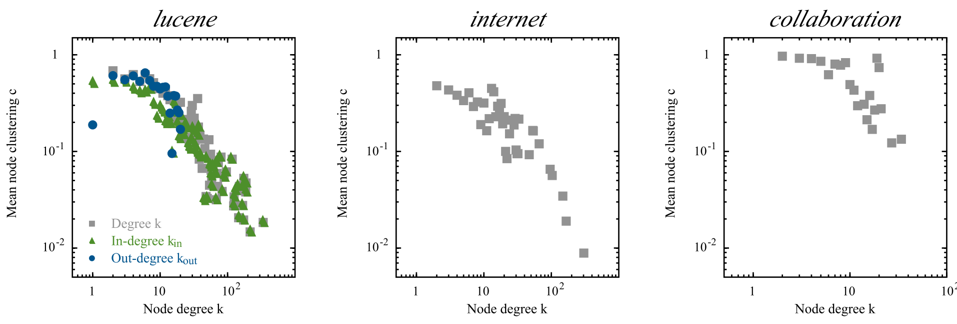


Fig. 5. Node clustering [57] profiles of larger networks (see also Table 4). Note degree biases introduced in the standard definition of clustering that imply low values for hubs, which is particularly apparent in degree disassortative networks (e.g., *lucene* and *internet* networks).

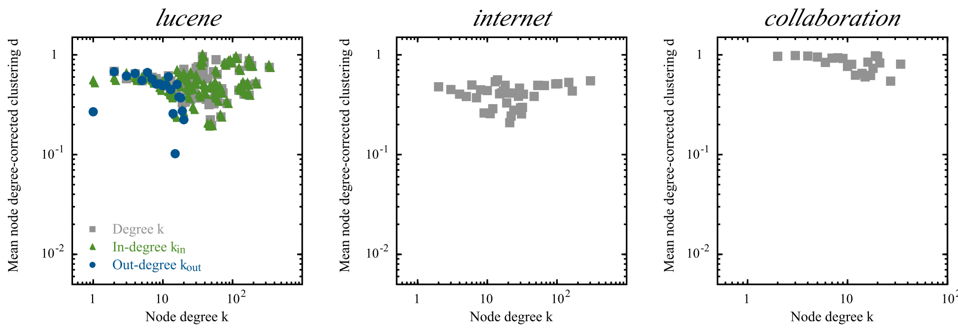


Fig. 6. Node degree-corrected clustering [43] profiles of larger networks (see also Table 4). Note that *lucene* software network reveals a sickle-shaped clustering profile most notably pronounced for out-degrees, which is absent in the case of the Internet and the *collaboration* network.

a power-law form $c \sim k^{-1}$ in the case of the Internet [56, 43]. Note that these biases are absent from the degree-corrected definition of clustering d (see Fig. 6), which thus provides somewhat more adequate measure of network transitivity.

Note also very peculiar sickle-shaped (degree-corrected) clustering profiles revealed for the software network (see, e.g., Fig. 5). This unique form is most notably pronounced in the case of out-degrees and is, at least in the undirected case, an artifact of dichotomous node degree mixing [19]. On the contrary, profiles of *internet* and *collaboration* networks show no particular scaling for degree-corrected clustering d (see Fig. 6), consistent with the analysis of node degree mixing in Sec. 3.2. Nevertheless, all networks considered here reveal clear degree-corrected clustering assortativity, which is throughly investigated in the following section.

Same as before, (degree-corrected) clustering profiles in software networks can be related to the intrinsic properties of the underlying software systems [46, 47]. While nodes that represent core classes of a software project commonly group together into dense neighborhoods with high clustering, nodes with lower clustering most often correspond to different implementations of the same functionality (see Fig. 14).

3.4. *Node degree-corrected clustering assortativity*

The present section explores node (degree-corrected) clustering mixing in different networks. For this purpose, we define clustering mixing coefficient r_c, $r_c \in [-1, 1]$, as

$$r_c = \frac{1}{2\sigma_c} \sum_{(i,j)\in L} (c_i - \langle c \rangle)(c_j - \langle c \rangle), \tag{4}$$

and similarly r_d for degree-corrected clustering coefficient. r_c and r_d are again just Pearson correlation coefficients of (degree-corrected) clustering at links' ends and are shown in Table 5. Due to degree biases in c (see Sec. 3.3), $r_c > 0$ in degree assortative networks (e.g., *collaboration* network), while $r_c < 0$ for networks that are disassortative by degree (e.g., *lucene* network). On the other hand, all networks show clear degree-corrected clustering assortativity with $r_d \gg 0$ (see also Fig. 7). Note also that correlations reflected in r_d are much stronger than in the case of degree mixing coefficients r-s (see Table 3). To the best of our knowledge, this distinctive property of real-world networks has not yet been reported in the literature.

According to Sec. 3.3, nodes in software networks have very different values of degree-corrected clustering d, which is not true for social networks or the Internet.

Table 5. Node clustering mixing coefficients of different networks.

Network	r_c	r_d
jbullet	−0.06	0.50
colt	−0.26	0.35
jung	−0.07	0.33
lucene	−0.40	0.50
internet	−0.23	0.26
collaboration	0.44	0.68

Note: Networks are reduced to simple undirected graphs.

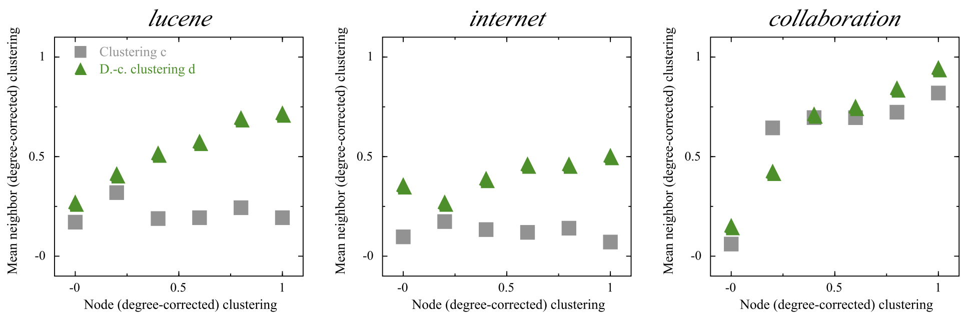


Fig. 7. Neighbor (degree-corrected) clustering plots of larger networks (see also Table 5). Note that all networks reveal a clear degree-corrected clustering [43] assortativity (e.g., *lucene* network), which is absent from the standard definition of clustering [57] (e.g., *internet* network).

Together with strong assortativity $r_d \gg 0$, this in fact implies entire connected parts or regions of nodes with rather similar d (e.g., very low or high). The latter can be clearly seen in Fig. 1, while, in the following section, we explain degree-corrected clustering assortativity, and dichotomous degree mixing observed in Sec. 3.2, through the presence of characteristic groups of nodes with common linking pattern [35]. More precisely, dense community-like groups occupy network regions with higher d and imply degree assortativity, while sparse module-like groups are found in regions with lower d and are responsible for degree disassortativity.

4. Group Structure of Software Networks

Node group structure of different networks is explored using a principled group extraction framework based on [49, 59]. The present section thus first introduces the framework and corresponding formalisms in Sec. 4.1, while Sec. 4.2 reports the characteristic group structure revealed in software and other networks. Last, Sec. 4.3 relates different types of groups to degree and clustering mixing observed in Sec. 3, which uniquely characterizes the structure of these networks.

4.1. *Node group extraction framework*

The formalism proposed in [49] defines network groups for the case of simple undirected graphs. Let S be a group of nodes and T a subset of nodes representing its characteristic linking pattern, $S, T \subseteq V$. Also, let $s = |S|$ and $t = |T|$. The node pattern T is defined thus to maximize the number of links between S and T, and minimize the number of links between S and T^C, while disregarding the links with both endpoints in S^C. Note that this simple formalism allows one to derive most types of groups commonly analyzed in the literature [13, 34] (Fig. 8).

For instance, communities [16], i.e., densely linked groups of nodes that are only sparsely linked between, are characterized by $S = T$. On the other hand, $S \cap T = \emptyset$ corresponds to groups of structurally equivalent [28] nodes denoted modules [48]. Communities and modules represent two extreme cases, with all other groups being the mixtures of the two [49]. For the analysis in the paper, we thus distinguish between three types of groups according to the following definitions.

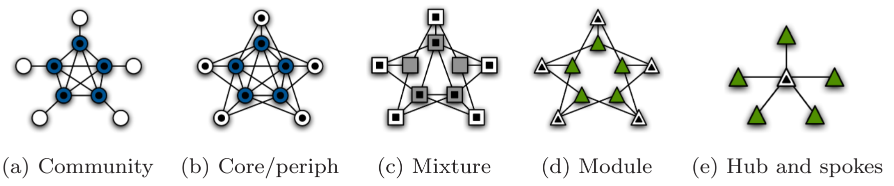


Fig. 8. Toy examples of different types of groups of nodes in real-world networks (see also text). (Groups S and corresponding patterns T are shown with filled and marked nodes, respectively.)

Definition 1. *Community* is a group of nodes S with $S = T$.

Definition 2. *Module* is a group of nodes S with $S \cap T = \emptyset$.

Definition 3. *Mixture* is a group of nodes S with $S \cap T \subset S, T$.

All these groups have been extensively analyzed in the past [42, 40, 13, 34]. Clear communities appear in different social and information networks [16, 10], while modules are most commonly found in the case the Internet, biological and technological networks [39, 48]. For consistency, we also consider two special cases.

Definition 4. *Core/periphery* structure is a mixture S with either $S \subset T$ or $T \subset S$.

Definition 5. *Hub and spokes* structure is a module S with $t = 1$.

According to the above definitions, one can in fact determine the type of some group S by considering Jaccard index [22] of S and T. We thus define a group type parameter τ [49], $\tau \in [0, 1]$, as

$$\tau(S,T) = \frac{|S \cap T|}{|S \cup T|}. \tag{5}$$

Communities have $\tau = 1$, whereas modules are indicated by $\tau = 0$. Mixtures correspond to groups with $0 < \tau < 1$. For the remaining of the paper, we refer to groups with $\tau \approx 1$ or $\tau \approx 0$ as community-like and module-like groups, respectively.

The framework presented below is based on a group criterion W [49], $W \in [0, 1]$.

$$W(S,T) = \mu(S,T)(1 - \mu(S,T)) \left(\frac{L(S,T)}{st} - \frac{L(S,T^C)}{s(n-t)} \right), \tag{6}$$

where $L(S,T)$ is the number of links between S and T, i.e., $L(S,T) = \sum_{(i,j) \in L} \delta(i \in S, j \in T)$, and $\mu(S,T)$ is the geometric mean of s and t normalized by the number of nodes n, $\mu \in [0, 1]$.

$$\mu(S,T) = \frac{2st}{n(s+t)}. \tag{7}$$

Note that W is an asymmetric criterion that favors the links between S and T, and penalizes for the links between S and T^C. Since the links with both endpoints in S^C are not considered, W is also a local criterion. We stress that, at least for the case $S = T$, criterion W has a natural interpretation in a wide class of different generative graph models [59] (e.g., block models [58]). Factor $\mu(1 - \mu)$ in Eq. (6) prevents from extracting either very small or large groups with, e.g., $s = 1$.

We next present the adopted group extraction framework [49, 59]. The framework extracts groups from the network sequentially, one by one, as follows. First, one finds group S and its corresponding pattern T that maximize criterion W using, e.g., tabu search [17] with varying initial conditions for S and T. At each step of the search, a single node is swapped in either S or T. Next, to extract the revealed group S from the network, one removes merely the links between S and T, and any node that might thus become isolated. The entire procedure is then repeated on the remaining network until criterion W is larger than the value expected under the same framework in a corresponding Erdös–Rényi random graph [11]. The latter is estimated by a simulation, thus, all groups reported in the remaining of the paper are statistically significant at the 1% level (see [59] for further details).

Note that the framework allows for overlapping [37], hierarchical [41], nested and other classes of groups commonly found in real-world networks. Nevertheless, it explicitly guards against extracting groups that are not statistically significant. We refer to the network structure remaining after the extraction as *background*.

4.2. *Characteristic node group structure*

Table 6 summarizes the basic properties of node groups extracted from different networks. Note that the mean group size $\langle s\rangle$ is somewhat comparable across software networks, where a characteristic group consists of around ten nodes. The mean pattern size $\langle t\rangle$ is slightly smaller, but still comparable to $\langle s\rangle$ (e.g., *jung* network). On the other hand, $\langle s\rangle \gg \langle t\rangle$ for the Internet, due to an abundance of hub and spokes-like modules. Since social networks are characterized by a pronounced community structure [36], expectedly, $\langle s\rangle \approx \langle t\rangle$ for *collaboration* network.

By examining the types of the revealed groups (see Table 6), one observes a very clear distinction between different networks. As already indicated above, *collaboration* network consists of almost only communities. On the contrary, 85% of the groups found in *internet* network are modules. Software networks, however, are characterized by communities, modules and different mixtures of these (e.g., *lucene* network). Thus, as already argued in the case of node mixing in Sec. 3, software networks represent a unique mixture of dense community-like structure of social networks and sparse module-like topology of the Internet. For a better comprehension, Fig. 9 shows most significant groups extracted from the networks.

Characteristic group structure of different networks is also reflected in the mean group parameter $\langle\tau\rangle$ (Table 7). Indeed, $\langle\tau\rangle$ is almost zero or one for *internet* and *collaboration* networks, respectively. For software networks, $\langle\tau\rangle$ is between 0.4 and 0.65, as discussed above. Table 7 also reports the proportion of links explained by the group structure, and the proportion of nodes included in the groups. Despite the

Table 6. Node groups and corresponding patterns extracted from different networks.

	Group			# ($\langle s\rangle$)			
Network	#	$\langle s\rangle$	$\langle t\rangle$	Community	Core/periphery	Mixture	Module
jbullet	14	9.0	8.4	5 (7.8)	1 (12.0)	6 (12.2)	2 (5.5)
colt	15	10.3	8.3	3 (8.3)	1 (13.0)	9 (12.6)	2 (6.5)
jung	30	8.7	7.8	18 (9.9)	1 (10.0)	5 (9.6)	6 (5.7)
lucene	123	12.1	7.9	55 (8.6)	2 (14.5)	27 (15.7)	39 (14.7)
internet	33	10.6	4.5	1 (4.0)	1 (29.0)	3 (19.0)	28 (9.6)
collaboration	160	5.6	5.6	143 (5.6)	0 (0.0)	12 (6.8)	5 (3.0)

Note: Networks are reduced to simple undirected graphs.

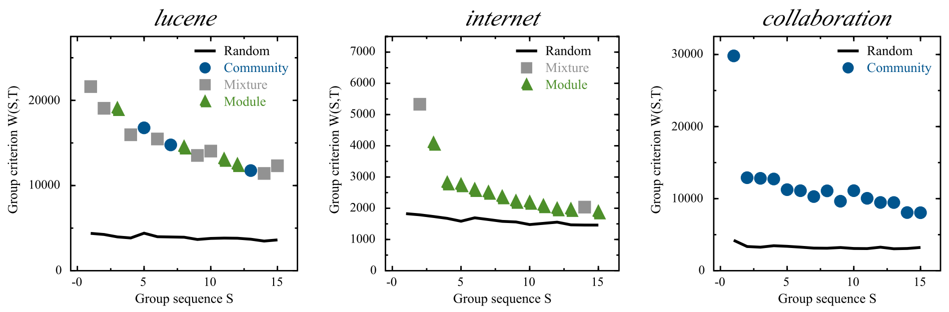


Fig. 9. Node group sequence extracted from larger networks (see also Table 6). Note that *lucene* software network contains communities, which are commonly found in social networks (e.g., *collaboration* network), modules like the Internet, and also different mixtures of these.

Table 7. Node group structure revealed in different networks (see also Table 6). Note that characteristic topology of different networks is well characterized by the mean group parameter $\langle\tau\rangle$.

		% Links (% nodes)[a]				
Network	Group $\langle\tau\rangle$	Community	Core/periph.	Mixture	Module	Background
jbullet	0.63	15 (22)	8 (7)	53 (42)	6 (7)	19 (66)
colt	0.41	7 (11)	5 (6)	69 (49)	4 (6)	15 (64)
jung	0.66	62 (51)	3 (3)	12 (16)	10 (11)	12 (44)
lucene	0.55	19 (25)	1 (2)	30 (24)	38 (34)	11 (49)
internet	0.08	0 (1)	12 (4)	13 (7)	34 (35)	41 (80)
collaboration	0.94	71 (47)	0 (0)	6 (5)	1 (1)	22 (47)

Note: Networks are reduced to simple undirected graphs.
[a]Nodes can be included in multiple overlapping groups.

fact that group structure provides a rather coarse-grained abstraction of a network, the reveled groups explain 80–90% of the links in software and social networks, and almost 60% for the Internet. Also, groups contain most of the nodes in the networks.

As already discussed in Sec. 3.3, different types of groups observed in software networks actually coincide with the intrinsic dynamics of the underlying software systems. More precisely, core classes of a software project commonly form dense inheritance hierarchies, while they also provide different convenience methods for transforming other core classes. Consequently, corresponding nodes in class dependency networks cluster together and form communities [46, 48] (see Fig. 14). Moreover, software projects commonly consist of classes that represent independent implementations of the same functionality (e.g., different group detection algorithms). By definition, these do not depend on each other. However, they do depend on a similar set of other classes. Hence, corresponding nodes in software networks aggregate together into module-like groups [48, 47] (see Fig. 14). Similarly as above, mixtures of nodes in software networks are often just an artifact of different programming principles and practical limitations of software systems.

Note also particularly module-like structure of *colt* network compared to other software networks (see $\langle\tau\rangle$ in Table 7). Since the network represents a software library for complex scientific and technical computing, high performance and scalability are of much greater importance than the system extensibility and future reusability. While the latter implies a modular design according to minimum-coupling and maximum-cohesion paradigm [45] and, consequently, a community-like structure of software networks [46], the former demands a great deal of code duplication, which in fact promotes module-like groups in software networks [48]. Equivalently, networks that correspond to software projects with particularly modular design reveal more community-like structure (e.g., *jung* network). Group structure of software networks thus reflects different programming principles and paradigms followed during project development, which could be used for software quality control.

Preliminary work on practical applications of network group detection in software engineering is described in Sec. 5, while, in the following section, we relate the characteristic group structure of software networks to previously observed dichotomous node degree mixing and degree-corrected clustering assortativity.

4.3. *Group degree and clustering mixing*

Section 3 shows that software networks are characterized by dichotomous node degree mixing that is assortative from the perspective of out-degrees, and disassortative from the perspective of in-degrees. Moreover, networks are composed of regions with rather similar clustering and reveal strong degree-corrected clustering assortativity. We have postulated a hypothesis that the observed structure is a consequence of different types of groups of nodes present in the networks. More precisely, software networks contain dense community-like groups in regions with higher clustering, which imply assortativity in the out-degree, and sparse module-like groups in regions with lower clustering, which promote disassortativity by in-degree, and different mixtures of these. As already discussed before, existence of different groups immediately explains also degree-corrected clustering assortativity.

We pursue the hypothesis by first investigating the regions of the networks occupied by different types of groups. Figure 10 shows group degree profiles that plot mean group parameter $\langle\tau\rangle$ against node degree k. These do not provide any clear insight into the structure of the networks, due to a rather extensive overlaps between the groups, i.e., both high and low degree nodes are included into different groups. On the other hand, group degree-corrected clustering profiles in Fig. 11 clearly show that software network indeed consists of module-like groups with $\tau \approx 0$ in sparse regions with low clustering $d \approx 0$ as hypothesized, while the plot reveals an expected increasing trend. Similarly, the network contains mostly community-like groups with $\tau \approx 1$ in dense regions with high clustering $d \approx 1$. However, the corresponding nodes are included also in overlapping module-like groups thus $\tau \approx 0.5$ (see Fig. 11). The same observations apply for social network and the Internet.

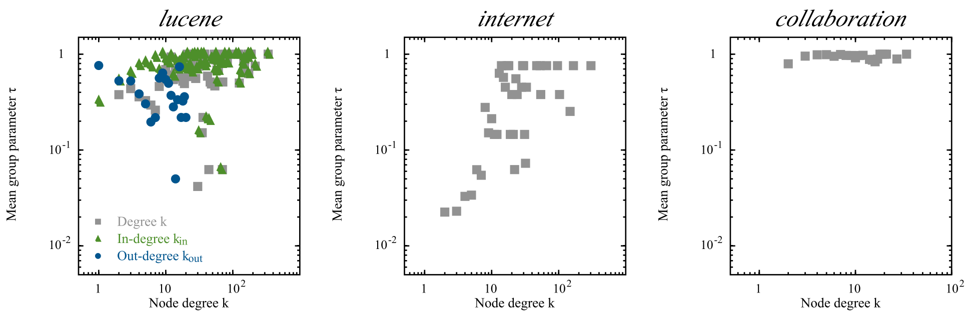


Fig. 10. Group degree profiles of larger networks that reveal no characteristic scaling.

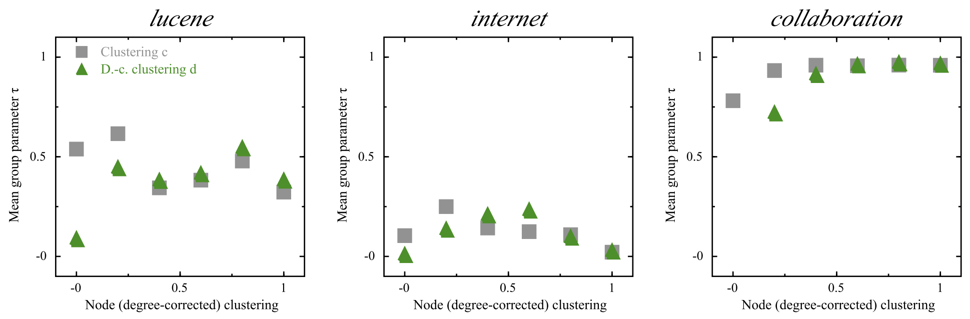


Fig. 11. Group (degree-corrected) clustering profiles of larger networks. Note that *lucene* software network consists of module-like groups with $\tau \approx 0$ in regions with $d \approx 0$ as the Internet and mostly community-like groups with $\tau \approx 1$ in regions with $d \approx 1$ as the *collaboration* network.

We next consider group degree and clustering mixing. For this purpose, we define group degree mixing coefficient $\tilde{r}$, $\tilde{r} \in [-1, 1]$, as

$$\tilde{r} = \frac{1}{\sigma_{\tilde{k}_S}\sigma_{\tilde{k}_T}} \sum_{S,T} (\tilde{k}_S - \langle \tilde{k}_S \rangle)(\tilde{k}_T - \langle \tilde{k}_T \rangle), \tag{8}$$

where $\tilde{k}_S$ is the degree of group S, i.e., $\tilde{k}_S = \sum_{i \in S} k_i / s$, and similarly for the pattern degree $\tilde{k}_T$. We further define also directed group degree mixing coefficients $\tilde{r}_{(\alpha,\beta)}$, $\alpha, \beta \in \{in, out\}$, and group clustering mixing coefficients $\tilde{r}_c$ and $\tilde{r}_d$, symmetrically as in Sec. 3. These provide an overview of degree and clustering mixing in regions covered by groups of nodes, and enable reasoning about the network structure implied by different types of groups.

Table 8 displays group mixing coefficients. Most evidently, almost all correlations observed in the case of node mixing are strictly enhanced (see Table 3). Social network is assortative by degree, while the Internet is degree disassortative. Software networks again reveal disassortativity in the in-degrees. However, in contrast to before, group structure in fact promotes assortativity by out-degree in all software networks except *colt* network, due to the reason given in Sec. 4.2. Figure 12 shows also group pattern connectivity plots. For software network, one can clearly

Table 8. Group degree and clustering mixing coefficients of different networks.

Network	$\tilde{r}$	$\tilde{r}_{(in,in)}$	$\tilde{r}_{(in,out)}$	$\tilde{r}_{(out,in)}$	$\tilde{r}_{(out,out)}$	$\tilde{r}_c$	$\tilde{r}_d$
jbullet	−0.02	−0.15	−0.01	−0.20	0.66	0.47	0.97
colt	−0.63	−0.60	−0.27	−0.63	−0.17	−0.59	0.76
jung	−0.32	−0.32	−0.12	−0.30	0.54	0.45	0.78
lucene	−0.16	−0.19	−0.12	−0.22	0.39	0.17	0.85
internet	−0.54	—	—	—	—	−0.37	0.37
collaboration	0.84	—	—	—	—	0.81	0.95

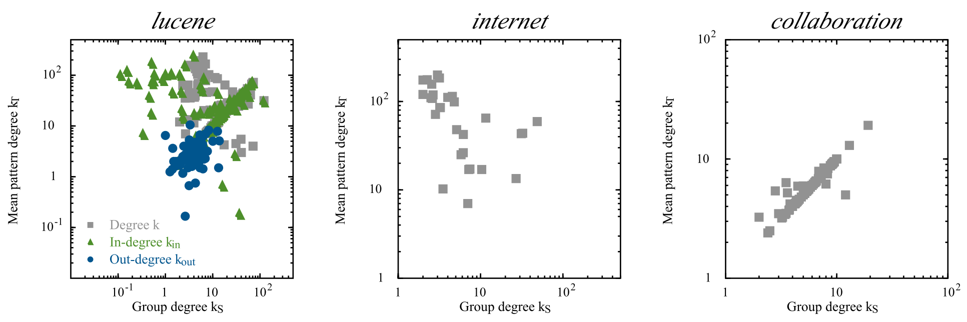


Fig. 12. Group pattern connectivity plots of larger networks (see also Table 8). Note that *lucene* software network reveals assortative mixing by out-degree as social networks (e.g., *collaboration* network) and disassortative mixing by in-degree as the Internet. While the former is an artifact of community-like groups, the latter is in fact a signature module-like groups.

observe an increasing trend in the case out-degrees, and also larger in-degrees, which is obviously an artifact of community-like groups, as in the case of social network. Otherwise, in-degree profile has a decreasing structure similar to that of the Internet, which signifies module-like groups. Thus, confirming the above hypothesis, group structure of software networks can indeed explain dichotomous degree mixing with module-like groups responsible for disassortativity, most notably seen for smaller in-degrees, and community-like groups promoting assortativity in the out-degrees.

It ought to be mentioned that the above relation between degree mixing and different groups of nodes can be justified theoretically. Since $S = T$ for communities, this implies degree assortativity, as long as the sizes of communities differ [36]. Also, for $s \not\approx t$, module-like groups should result in degree disassortativity [48]. Finally, according to discussion in Sec. 4.2, modules or communities are best pronounced through the out-degrees and in-degrees of nodes, respectively.

Table 8 also reports group clustering mixing coefficients. As before, $\tilde{r}_c < 0$ in some degree disassortative networks, due to the biases introduced in clustering c (see Sec. 3.3). Nevertheless, degree-corrected clustering mixing $\tilde{r}_d$ signifies extremely assortative structure with correlations between 0.75 and 0.95 for software and social networks (see also Fig. 13). Presence of clear groups of nodes thus indeed implies degree-corrected clustering assortativity, while the value of $\tilde{r}_d$ can be related to the

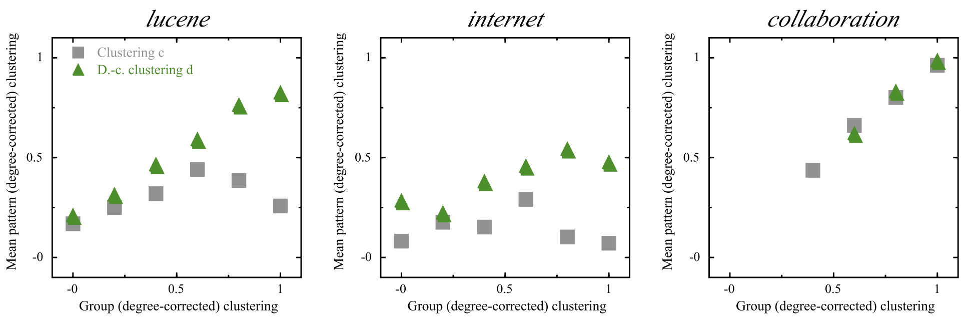


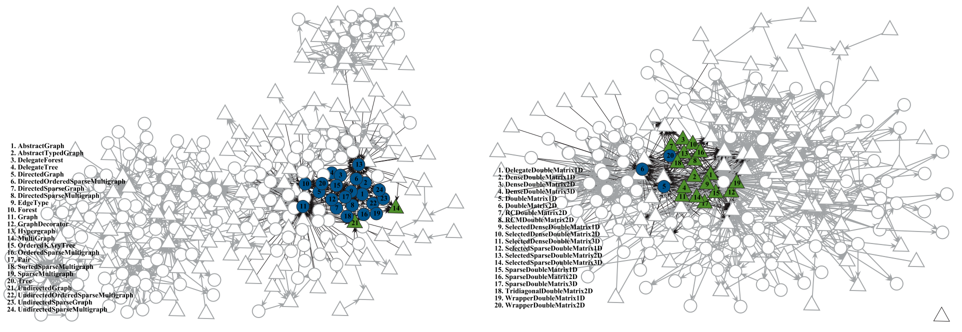
Fig. 13. Group pattern (degree-corrected) clustering plots of larger networks (see also Table 8). Note that networks reveal extremely clear group degree-corrected clustering [43] assortativity (e.g., *lucene* and *collaboration* network), which is an indication of a well pronounced group structure.

quality of network group structure. For example, in the case of the Internet, which has least clear group structure (see Sec. 4.2), $\tilde{r}_d$ is only 0.37.

In summary, characteristic groups of nodes provide an important insight into the dynamics of complex networks and can, at least to some extent, explain the unique structure of software networks (i.e., degree and clustering mixing). There is of course no reason why the same principles should not apply to other real-world networks, directed or undirected, which will be thoroughly explored in future work.

5. Applications in Software Engineering

The present section describes preliminary work on practical applications of network group detection in software engineering. As already discussed before, groups of nodes in software dependency networks coincide with the intrinsic properties of the underlying software systems. For instance, Fig. 14 shows the most significant groups


1. AbstractGraph
2. AbstractTypedGraph
3. DelegateForest
4. DelegateTree
5. DirectedGraph
6. DirectedOrderedSparseMultigraph
7. DirectedSparseGraph
8. DirectedSparseMultigraph
9. EdgeType
10. Forest
11. Graph
12. GraphDecorator
13. Hypergraph
14. MultiGraph
15. OrderedKAryTree
16. OrderedSparseMultigraph
17. Pair
18. SortedSparseMultigraph
19. SparseMultigraph
20. Tree
21. UndirectedGraph
22. UndirectedOrderedSparseMultigraph
23. UndirectedSparseGraph
24. UndirectedSparseMultigraph
1. DelegateDoubleMatrix1D
2. DenseDoubleMatrix1D
3. DenseDoubleMatrix2D
4. DenseDoubleMatrix3D
5. DoubleMatrix1D
6. DoubleMatrix2D
7. RCDoubleMatrix2D
8. RCMDoubleMatrix2D
9. SelectedDenseDoubleMatrix1D
10. SelectedDenseDoubleMatrix2D
11. SelectedDenseDoubleMatrix3D
12. SelectedSparseDoubleMatrix1D
13. SelectedSparseDoubleMatrix2D
14. SelectedSparseDoubleMatrix3D
15. SparseDoubleMatrix1D
16. SparseDoubleMatrix2D
17. SparseDoubleMatrix3D
18. TridiagonalDoubleMatrix2D
19. WrapperDoubleMatrix1D
20. WrapperDoubleMatrix2D

(a) Community in *jung* network ($\tau = 1$) (b) Module-like group in *colt* network ($\tau = 0.06$)

Fig. 14. Most significant groups of nodes extracted from different software networks (see also Table 6). The groups correspond to (a) core classes of the software project and (b) different implementations of classes with the same functionality. (Nodes with degree-corrected clustering [43] above or below the mean are shown as circles and triangles, respectively.)

revealed in *jung* and *colt* networks. In the case of the former, the best group is a community that corresponds to core classes of the project, as predicted in Sec. 4.2. Since the network represents a framework for graph and network analysis, these are actually different graphs, multigraphs, hypergraphs and trees. Note that the revealed group is not only very clear, but also rather exhaustive.

On the other hand, the most significant group in *colt* network, which represents a software library for high-performance scientific computing, is module-like and contains different implementations of matrices (e.g., dense, sparse or wrapped). Recall that the latter is consistent with the rationale behind the existence of modules in software networks given in Sec. 4.2. Similarly as above, the group is indeed transparent, while the identifiers of the corresponding software classes are extremely consistent with each other [see Fig. 14(b)]. Thus, one can in fact derive templates for class identifiers (e.g., by mining common textual patterns [1]) and unique class dependencies on the level of groups of nodes in a software network (i.e., by analyzing corresponding node patterns). These can be adopted in future project development, in order to maintain a high consistency of a software system, to reduce code duplication issues and other. Furthermore, one can also predict the package of a class.

Classes of object-oriented software systems are organized into software packages that form a complex hierarchy. Each class is a member of exactly one package, whereas the classes can reside also in the inner nodes of the package hierarchy. For example, the group of nodes shown in Fig. 14(a) consists mostly of classes in `edu.uci.ics.jung.graph` package, while the group in Fig. 14(b) represents classes in `cern.colt.matrix.impl` package. To predict the package of some class given the group structure of the software network, we investigate the classes, whose nodes are residing in the same network groups as the concerned one. These classes are then weighted according to the Jaccard similarity [22] between the corresponding nodes' neighborhoods and their packages are taken as the candidates for the prediction. We select the most frequent package with respect to weights, while ties are broken uniformly at random (see [46, 47] for details). Note that, instead of considering nodes within the same network groups, one can of course examine merely nodes' neighbors or the entire network. For comparison, we also report the performance of a classifier that predicts the most frequent (i.e., majority) package within the software system for each class and a random classifier. However, the adoption of some more sophisticated approaches like deep belief nets [20] or structured support vector machine [50] would inevitably require the identification of learning features.

Table 9 shows classification accuracy for software package prediction. Observe that the accuracy for the strategy based on network groups is around 75% in all cases except for the larger *lucene* network. We stress that the latter is an impressive result. Indeed, the task at hand represents an extremely difficult classification problem due to a large number of possible classifications, while this number is else two or three in most practical applications (see performance of the baseline classifiers). Note also that the strategy based on nodes' neighbors performs very well in *jbullet* and *jung* networks with more community-like groups (see $\langle\tau\rangle$ in Table 7), since the groups

Table 9. Classification accuracy of software package prediction based on the node's neighbors Γ or groups S, or the entire network N (see text for details).

			(%)				
Network	# Classes[a]	# Packages	Γ	S	N	Majority	Random
jbullet	107	11	72.0	**75.7**	64.5	28.0	8.6
colt	154	16	58.4	**73.4**	55.2	22.7	5.9
jung	237	31	72.2	**74.2**	65.0	11.4	3.3
lucene	1335	178	47.1	**49.2**	43.7	6.4	0.6

Note: Results are averages over 100 runs.
[a]Analysis is reduced to nodes included in network groups.

Table 10. Classification accuracy of high-level software package prediction based on the node's neighbors Γ or groups S, or the entire network N (see text for details).

			(%)				
Network	# Classes[a]	# Packages	Γ	S	N	Majority	Random
jbullet	107	5	**84.6**	**85.0**	78.5	64.5	20.4
colt	154	10	**86.4**	83.8	69.5	39.0	9.7
jung	237	5	89.1	**90.5**	**91.1**	44.3	20.3
lucene	1335	15	85.5	**90.8**	85.0	28.2	6.6

Note: Results are averages over 100 runs.
[a]Analysis is reduced to nodes included in network groups.

well coincide with nodes' neighborhoods. On the other hand, the neighbors are in fact different from one another in *colt* network with more module-like groups (see Sec. 4), which significantly decreases the performance.

Table 10 shows also the accuracy for high-level software package prediction problem, where we consider only the packages at the topmost level of the package hierarchy. For *jung* network, these are `graph`, `algorithms`, `io`, `visualization` and `visualization3d` (prefix `edu.uci.ics.jung` is omitted). Again, the strategy based on network groups performs particularly well with classification accuracy around 85–90%. Besides, the strategy based on nodes' neighbors, and also the network-based strategy for *jung* network, obtains surprisingly high results, which further justifies the construction of software dependency networks (see Sec. 2).

Thus, characteristic group structure of software networks can indeed be exploited to quite accurately infer the package hierarchy of software systems [46, 47]. This has numerous applications. For instance, the framework can be used to predict packages of new classes introduced into an unknown software project or even the programming language itself, to detect possibly duplicated classes, or for merging classes across different software packages or libraries (one by one). Such tasks would else demand significant manual labor, especially for large and complex software systems. Furthermore, network group detection can be adopted for software project refactoring, in order to derive either more modular or more functional software package hierarchy [47, 48] (i.e., community-like and module-like, respectively).

Table 11. Classification accuracy of class prediction for *jung* software network based on the node's neighbors Γ or groups S, or the entire network N (see text for details).

		(%)				
Prediction	# Categories	Γ	S	N	*Majority*	*Random*
Class type	2	65.0	**85.2**	**84.8**	**84.4**	49.9
Class version	9	67.7	**72.8**	66.2	44.3	11.2
Class author	11	**71.6**	**71.0**	**70.9**	44.3	9.2

Note: Results are averages over 100 runs.

As shown below, characteristic groups in software networks can also be used to infer the name of the developer that implemented a particular class, the exact version at which it was introduced into the project or its type (i.e., class or interface). However, as this information was largely unavailable or could not be obtained automatically for the software projects considered, we only report the results for *jung* network. The prediction else proceeds exactly the same as before, while the classes with unknown version or author information are grouped into a single category.

Table 11 shows the classification accuracy for different software prediction problems. For class type prediction, the strategy based on network groups performs only slightly better than the baseline approach that classifies all software classes into the same category. On the other hand, the performance is significantly improved in the case of class version and author prediction problems with accuracy over 70%. This is not very surprising, since classes with the same functionality that appear as different groups in software networks are commonly introduced within the same version of the software project and implemented by the same developer.

Furthermore, according to Sec. 4.2, the quality of network group structure reflects different programming principles and paradigms. Since this can be measured by degree-corrected group clustering mixing (see Sec. 4.3), the latter enables different applications in software development and quality control.

6. Conclusion and Future Work

The present paper rigorously analyzes the structure of complex software networks. These can be seen as an interplay between a dense structure of social networks and a sparse topology of the Internet. In particular, we show that software networks reveal characteristic node group structure, which consists of dense communities, sparse module-like groups and also different mixtures of these. Communities imply assortative mixing by degree, whereas just the opposite holds for the modules. Thus, software networks reveal dichotomous degree mixing that is assortative in the out-degrees and disassortative in the in-degrees. Furthermore, communities appear in denser regions with higher clustering, while most pronounced modules occupy sparse regions with very low clustering. The latter in fact promotes degree-corrected clustering assortativity, which is observed in all of the networks analyzed.

Besides, the group structure of software networks also coincides with the intrinsic properties of the underlying software systems. The paper thus includes some preliminary work on practical applications of network group detection in software engineering. Nevertheless, their true practical value in real scenarios remains somewhat unclear and will be more throughly investigated in the future.

The study of differences between software and social networks, and the Internet, reveals notably distinct network topologies that are most likely governed by different phenomena. We stress that dichotomous node degree mixing has not yet been observed in the case of directed networks. Furthermore, preliminary results show that the existing graph models do not produce degree-corrected clustering assortativity of real-world networks. The latter will be the main focus of our future work.

Additionally, the paper implies several other prominent directions for future research. First, the observed node mixing and group structure might also apply to different software and other real-world networks. Among these, various information networks seem most promising. Next, characteristic group structure revealed for software networks might be further related to other properties, e.g., self-similarity [4] or hierarchical structure [55]. Last, although we provide some rationale for the presence of groups in software networks, a generative graph model is still an open issue.

Acknowledgments

This work has been supported in part by the Slovenian Research Agency Program No. P2-0359, by the Slovenian Ministry of Education, Science and Sport Grant No. 430-168/2013/91, and by the European Union, European Social Fund.

References

[1] Baeza-Yates, R. and Ribeiro-Neto, B., *Modern Information Retreival* (Addison-Wesley, Harlow, UK, 1999).

[2] Barabási, A. L. and Albert, R., Emergence of scaling in random networks, *Science* **286** (1999) 509–512.

[3] Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Visser, M., Melton, H. and Tempero, E., Understanding the shape of java software, in *Proc. ACM Int. Conf. Object-Oriented Programming, Systems, Languages, and Applications* (Portland, OR, USA, 2006), pp. 397–412.

[4] Blagus, N., Šubelj, L. and Bajec, M., Self-similar scaling of density in complex real-world networks, *Physica A* **391** (2012) 2794–2802.

[5] Cai, K.-Y. and Yin, B.-B., Software execution processes as an evolving complex network, *Inform. Sci.* **179** (2009) 1903–1928.

[6] Chidamber, S. and Kemerer, C., A metrics suite for object oriented design, *IEEE Trans. Softw. Eng.* **20** (1994) 476–493.

[7] Clauset, A., Shalizi, C. R. and Newman, M. E. J., Power-law distributions in empirical data, *SIAM Rev.* **51** (2009) 661–703.

[8] Concas, G., Locci, M. F., Marchesi, M., Pinna, S. and Turnu, I., Fractal dimension in software networks, *Europhys. Lett.* **76** (2006) 1221–1227.

[9] Concas, G., Marchesi, M., Pinna, S. and Serra, N., Power-laws in a large object-oriented software system, *IEEE Trans. Softw. Eng.* **33** (2007) 687–708.

[10] Danon, L., Díaz-Guilera, A., Duch, J. and Arenas, A., Comparing community structure identification, *J. Stat. Mech.* (2005) P09008.

[11] Erdős, P. and Rényi, A., On random graphs i, *Publ. Math. Debrecen* **6** (1959) 290–297.

[12] Fortuna, M. A., Bonachela, J. A. and Levin, S. A., Evolution of a modular software network, *Proc. Natl. Acad. Sci. USA* **108** (2011) 19985–19989.

[13] Fortunato, S., Community detection in graphs, *Phys. Rep.* **486** (2010) 75–174.

[14] Foster, J. G., Foster, D. V., Grassberger, P. and Paczuski, M., Edge direction and the structure of networks, *Proc. Natl. Acad. Sci. USA* **107** (2010) 10815–10820.

[15] Gao, Y., Xu, G., Yang, Y., Liu, J. and Guo, S., Disassortativity and degree distribution of software coupling networks in object-oriented software systems, in *Proc. IEEE Int. Conf. Progress in Informatics and Computing* (Shanghai, China, 2010), pp. 1000–1004.

[16] Girvan, M. and Newman, M. E. J., Community structure in social and biological networks, *Proc. Natl. Acad. Sci. USA* **99** (2002) 7821–7826.

[17] Glover, F., Tabu search, *ORSA J. Comput.* **1** (1989) 190–206.

[18] Han, J.-D. J., Bertin, N., Hao, T., Goldberg, D. S., Berriz, G. F., Zhang, L. V., Dupuy, D., Walhout, A. J. M., Cusick, M. E., Roth, F. P. and Vidal, M., Evidence for dynamically organized modularity in the yeast protein-protein interaction network, *Nature* **430** (2004) 88–93.

[19] Hao, D. and Li, C., The dichotomy in degree correlation of biological networks, *PLoS One* **6** (2011) e28322.

[20] Hinton, G. E., Osindero, S. and Teh, Y.-W., A fast learning algorithm for deep belief nets, *Neural Comput.* **18** (2006) 1527–1554.

[21] Hyland-Wood, D., Carrington, D. and Kaplan, S., Scale-free nature of java software package, class and method collaboration graphs, in *Proc. Int. Symp. Empirical Software Engineering* (Rio de Janeiro, Brazil, 2006), pp. 1–10.

[22] Jaccard, P., Étude comparative de la distribution florale dans une portion des alpes et des jura, *Bull. Soc. Vaud. Sci. Nat.* **37** (1901) 547–579.

[23] Kohring, G. A., Complex dependencies in large software systems, *Adv. Complex Syst.* **12** (2009) 565–581.

[24] LaBelle, N. and Wallingford, E., Inter-package dependency networks in open-source software, in *Proc. Int. Conf. Complex Systems* (Boston, MA, USA, 2006), pp. 1–8.

[25] Lancichinetti, A., Kivela, M., Saramaki, J. and Fortunato, S., Characterizing the community structure of complex networks, *PLoS One* **5** (2010) e11976.

[26] Leskovec, J., Kleinberg, J. and Faloutsos, C., Graphs over time: Densification laws, shrinking diameters and possible explanations, in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining* (Chicago, IL, USA, 2005), pp. 177–187.

[27] Li, H., Zhao, H., Cai, W., Xu, J.-Q. and Ai, J., A modular attachment mechanism for software network evolution, *Physica A* **392** (2013) 2025–2037.

[28] Lorrain, F. and White, H. C., Structural equivalence of individuals in social networks, *J. Math. Sociol.* **1** (1971) 49–80.

[29] Maslov, S. and Sneppen, K., Specificity and stability in topology of protein networks, *Science* **296** (2002) 910–913.

[30] Myers, C. R., Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, *Phys. Rev. E* **68** (2003) 046116.

[31] Newman, M. E. J., Assortative mixing in networks, *Phys. Rev. Lett.* **89** (2002) 208701.

[32] Newman, M. E. J., Mixing patterns in networks, *Phys. Rev. E* **67** (2003) 026126.

[33] Newman, M. E. J., Finding community structure in networks using the eigenvectors of matrices, *Phys. Rev. E* **74** (2006) 036104.
[34] Newman, M. E. J., Communities, modules and large-scale structure in networks, *Nat. Phys.* **8** (2012) 25–31.
[35] Newman, M. E. J. and Leicht, E. A., Mixture models and exploratory analysis in networks, *Proc. Natl. Acad. Sci. USA* **104** (2007) 9564.
[36] Newman, M. E. J. and Park, J., Why social networks are different from other types of networks, *Phys. Rev. E* **68** (2003) 036122.
[37] Palla, G., Derényi, I., Farkas, I. and Vicsek, T., Uncovering the overlapping community structure of complex networks in nature and society, *Nature* **435** (2005) 814–818.
[38] Pastor-Satorras, R., Vázquez, A. and Vespignani, A., Dynamical and correlation properties of the internet, *Phys. Rev. Lett.* **87** (2001) 258701.
[39] Pinkert, S., Schultz, J. and Reichardt, J., Protein interaction networks: More than mere modules, *PLoS Comput. Biol.* **6** (2010) e1000659.
[40] Porter, M. A., Onnela, J.-P. and Mucha, P. J., Communities in networks, *Not. Am. Math. Soc.* **56** (2009) 1082–1097.
[41] Ravasz, E., Somera, A. L., Mongru, D. A., Oltvai, Z. N. and Barabási, A. L., Hierarchical organization of modularity in metabolic networks, *Science* **297** (2002) 1551–1555.
[42] Schaeffer, S. E., Graph clustering, *Comput. Sci. Rev.* **1** (2007) 27–64.
[43] Soffer, S. N. and Vázquez, A., Network clustering coefficient without degree-correlation biases, *Phys. Rev. E* **71** (2005) 057101.
[44] Solé, R. V., Ferrer-Cancho, R., Montoya, J. M. and Valverde, S., Selection, tinkering, and emergence in complex networks, *J. Complexity* **8** (2003) 20–33.
[45] Stevens, W. P., Myers, G. J. and Constantive, L. L., Structured design, *IBM Syst. J.* **38** (1999) 231–256.
[46] Šubelj, L. and Bajec, M., Community structure of complex software systems: Analysis and applications, *Physica A* **390** (2011) 2968–2975.
[47] Šubelj, L. and Bajec, M., Software systems through complex networks science: Review, analysis and applications, in *Proc. KDD Workshop on Software Mining* (Beijing, China, 2012), pp. 9–16.
[48] Šubelj, L. and Bajec, M., Ubiquitousness of link-density and link-pattern communities in real-world networks, *Eur. Phys. J. B* **85** (2012) 32.
[49] Šubelj, L., Blagus, N. and Bajec, M., Group extraction for real-world networks: The case of communities, modules, and hubs and spokes, in *Proc. Int. Conf. Network Science* (Copenhagen, Denmark, 2013), pp. 152–153.
[50] Tsochantaridis, I., Joachims, T., Hofmann, T. and Altun, Y., Large margin methods for structured and interdependent output variables, *J. Mach. Learn. Res.* **6** (2005) 1453–1484.
[51] Turnu, I., Concas, G., Marchesi, M. and Tonelli, R., The fractal dimension of software networks as a global quality metric, *Inform. Sci.* **245** (2013) 290–303.
[52] Valverde, S., Cancho, R. F. and Solé, R. V., Scale-free networks from optimal design, *Europhys. Lett.* **60** (2002) 512–517.
[53] Valverde, S. and Solé, R. V., Logarithmic growth dynamics in software networks, *Europhys. Lett.* **72** (2005) 858–864.
[54] Valverde, S. and Solé, R. V., Network motifs in computational graphs: A case study in software architecture, *Phys. Rev. E* **72** (2005) 026107.
[55] Valverde, S. and Solé, R. V., Hierarchical small worlds in software architecture, *Dyn. Cont. Dis. Ser. B* **14** (2007) 1–11.

[56] Vázquez, A., Pastor-Satorras, R. and Vespignani, A., Large-scale topological and dynamical properties of the internet, *Phys. Rev. E* **65** (2002) 066130.

[57] Watts, D. J. and Strogatz, S. H., Collective dynamics of "small-world" networks, *Nature* **393** (1998) 440–442.

[58] White, H. C., Boorman, S. A. and Breiger, R. L., Social structure from multiple networks: Block models of roles and positions, *Am. J. Sociol.* **81** (1976) 730–779.

[59] Zhao, Y., Levina, E. and Zhu, J., Community extraction for social networks, *Proc. Natl. Acad. Sci. USA* **108** (2011) 7321–7326.