Community Detection and Analysis of Community Evolution in Apache Ant Class Collaboration Networks

Miloš Savić
svc@dmi.uns.ac.rs

Miloš Radovanović
radacha@dmi.uns.ac.rs

Mirjana Ivanović
mira@dmi.uns.ac.rs

Department of Mathematics and Informatics, Faculty of Science, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia

ABSTRACT

In this paper we investigate community detection algorithms applied to class collaboration networks (CCNs) that represent class dependencies of 21 consecutive versions of the Apache Ant software system. Four community detection techniques, Girvan-Newman (GN), Greedy Modularity Optimization (GMO), Walktrap and Label Propagation (LP), are used to compute community partitions. Obtained community structures are evaluated using community quality metrics (inter- and intra-cluster density, conductance and expansion) and compared to package structures of analyzed software. In order to investigate evolutionary stability of community detection methods, we designed an algorithm for tracking evolving communities. For LP and GMO, algorithms that produce partitions with higher values of normalized modularity score compared to GN and Walktrap, we noticed an evolutionary degeneracy – LP and GMO are extremely sensitive to small evolutionary changes in CCN structure. Walktrap shows the best performance considering community quality, evolutionary stability and comparison with actual class groupings into packages. Coarse-grained descriptions (CGD) of CCNs are constructed from Walktrap partitions and analyzed. Results suggest that CCNs have modular structure that cannot be considered as hierarchical, due to the existence of large strongly connected components in CGDs.

Categories and Subject Descriptors

D.2.8 [**Metrics**]: Product metrics, Software science; E.1 [**Data structures**]: Graphs and networks

General Terms

Algorithms, Measurement, Experimentation.

Keywords

Community detection, class collaboration network, community evolution, evolutionary degeneracy.

1. INTRODUCTION

A large computer program written in an object-oriented (OO) programming language is typically divided into a set of classes and interfaces. Classes and interfaces can collaborate in various ways: a class can implement an interface or extend the functionality of another class, define a member variable whose type is another class; it may use another class as the type of an input parameter or return value of one of its methods, instantiate objects of another class, etc. In all these cases we say that a class references another class. An OO software system can be represented as a graph in which classes defined in the system can be viewed as nodes and references between them as links. This kind of graph is known as a *class collaboration network* (CCN). CCNs are simplified class diagrams, a notion extensively used in object-oriented design and analysis, because they preserve only the existence of relations between classes and discard other types of information about nodes (classes) and links (OO references) presented in class diagrams. Such abstraction enables the study of complex software systems as regular networks under the framework of the complex network theory. Recent progress in complex network theory resulted with network measures, statistical analysis techniques, evolutionary principles and mathematical models which can reveal and explain frequently observed macroscopic properties of complex networks such as the small-world property, power-law degree distribution (scale-free property), high values of local clustering coefficients comparing to random graphs, "robust yet fragile" property and highly modular or community structure [1, 2].

Informally speaking, a community, cluster or module is a cohesive set of nodes in a network that are more highly connected to each other than to the rest of the network. Community structure is a typical feature of social networks (such as online social networks and scientific collaboration networks) but is also characteristic to other types of complex networks. For example, tightly connected groups of nodes in the WWW often correspond to pages dealing with the same topic (as was shown in [3]), while in cellular and genetic networks are related to functional modules [2]. Uncovering communities helps us to understand the structure of the network and to draw a readable map (coarse-grained description, network of communities) of extremely large networks.

For software systems, one of the essential design principles is that of low coupling and high cohesion between software components or modules. In a highly modular OO software system, each module is a set of functionally related classes which are highly cohesive. Coupling between modules should be loose in order to preserve their autonomy, to restrict change propagation and to make modules reusable. Therefore, the low coupling and high cohesion principle itself promotes the emergence of communities in class collaboration networks representing well designed software systems.

An important advance in community detection was made by Girvan and Newman [4], who introduced a measure for the quality of a partition of a network called modularity. Modularity is defined as $Q = \sum_{s=1}^{m} \left[\frac{l_s}{L} - \left(\frac{d_s}{2L} \right)^2 \right]$, where the sum goes over the m clusters of the partition, l_s is the number of links inside community s, L is the total number of links in the network, and d_s represents the total degree of the nodes in community s. The first term of summand in previous equation is the fraction of links inside cluster s. The second term represents expected fraction of links inside the cluster in the randomized network of the same size and same degree sequence (network sampled using the configuration model [5]). If the first term is much larger than the second, then there are more links inside the cluster that one would expect by random chance. The comparison with the configuration model leads to a modularity-based definition of a community (or cluster): a sub-graph s is a cluster if $\frac{l_s}{L} - \left(\frac{d_s}{2L} \right)^2 > 0$.

Although widely used, the modularity measure has two weaknesses. Fortunato and Barthélemy [6] showed that the modularity measure has an intrinsic scale that depends on the total number of links in the network (the resolution limit problem). Communities that are smaller than this scale cannot be detected through modularity maximization methods, even in the extreme case when they are complete sub-networks connected by single bridges. Good et al. [7] showed that there are typically an exponential number of structurally diverse alternative partitions with modularity scores very close to the maximum (the degeneracy problem). However, modularity is still commonly used for evaluation of network community structure in community detection algorithms, mainly due to the lack of other measures [8].

In this paper, we investigate community detection techniques applied to an evolutionary sequence of class collaboration networks representing Apache Ant in 21 versions, and observe the properties of obtained communities. Four community detection techniques are considered: Girwan-Newman (GN), Greedy Modularity Optimization (GMO), Walktrap and Label Propagation (LP).

The rest of paper is structured as follows. In Section 2 related work, motivation for this study and main contributions are presented. The following section describes applied techniques for community detection and an algorithm for tracking evolving communities. In Section 4 obtained results are presented and discussed. Finally, in Section 5 we give conclusions and directions for future work.

2. RELATED WORK

In recent years, studies have shown that many natural and artificial complex systems can be investigated under the framework of complex network theory [1, 2]. Among others, networks representing dependencies between entities defined in a software system were identified as an important class of complex networks. Different authors [9–12] observed that software networks exhibit scale-free (or partial scale-free [13]) and small-world characteristics. On the other hand, community structure of software networks has not yet been thoroughly investigated. In this section we present articles that deal with community detection in class collaboration networks associated with software systems written in Java.

Dietrich et al. [14] developed BARIO, an Eclipse plugin that can detect and visualize clusters in CCNs associated with Java programs. BARIO uses the GN algorithm to compute community partitions. For four Java programs, the authors investigated how APC (the average number of packages per cluster) and ACP (the average number of clusters per package) change with the number of iterations in the GN algorithm.

Šubelj and Bajec [8] analyzed the community structure of CCNs associated with six Java software systems (JUnit, JavaMail, Flamingo, Jung, Colt and JDK). To reveal the community structure of each network, the authors employed GN, GMO and LP. Obtained partitions possess high values of modularity (0.55–0.75), but identified communities do not exactly correspond to the modular structure defined by the package specification. In this work we include the Walktrap community detection algorithm that was not considered in [8]. As the main contribution we will show that this algorithm gives the best performance when clustering Apache Ant considering at the same time three aspects: community structure quality, evolutionary stability and correspondence with the groupings of classes defined by Apache Ant packages.

Paymal et al. [15] investigated the community structure in CCNs extracted from six consecutive versions of JHotDraw software using the GMO technique. Authors observed that two largest communities contain 50% or more of all nodes in each version and that those two communities have continuous and stable growth during software evolution. Comparing to [9], we examine a software system with higher number of versions and apply more than one community detection technique. As we will show, using the Apache Ant as the case study, although different community detection algorithms produce community partitions with similar values of the normalized modularity score (which is stabile during software evolution), they drastically differ in the number of resulting communities and evolutionary stability. This means that more than one community detection approach always needs to be employed in practice, and the best partitioning chosen using multi-view criteria.

3. METHODOLOGY

Class collaboration networks associated with 21 consecutive versions of Apache Ant (from version 1.1.0 to 1.8.2) were extracted from source code using a software tool called Yaccne [12]. Yaccne contains a subprogram called Diff that finds the differences between two networks where the second evolves from the first one by adding new nodes, deleting some nodes, adding new links and/or deleting some links.

For community detection in Apache Ant class collaboration networks, we developed C software based on the *igraph* library [16] that contains implementations of four community detection algorithms investigated in this paper: Girwan-Newman (GN), Greedy Modularity Optimization (GMO), Walktrap and Label Propagation (LP).

The GN algorithm [4] is based on the edge betweenness centrality measure. The betweenness of edge e is defined as the fraction of shortest paths between all pairs of nodes passing through e. Since the edges that lie between communities are expected to have high values of betweenness, a clustering dendrogram can be obtained in the divisive manner by recursively removing edges with the highest value of betweenness. When all edges are removed, the dendrogram is cut at the level with the highest value of the modularity score.

GMO [17] starts with n clusters each containing a single node, where n is the size of network. At each iteration, the algorithm computes or updates the previous value of the variation ΔQ of the modularity score obtained by merging any two communities. The merger which maximally increases (or minimally decreases) the modularity score is chosen and the merge is performed. Thus, a clustering dendrogram is made in the agglomerative manner.

The main idea of the Walktrap community detection algorithm [18] is that random walks would be trap into dense sub-networks thanks to the high density of links in the community. The algorithm uses a distance measure between nodes that is calculated from the probabilities that the random walker moves from a node to another in k steps. Nodes are then grouped into communities through an agglomerative hierarchical clustering technique based on Ward's method. For each class collaboration network we performed d runs of Walktrap varying the value of k from 1 to d, where d is the diameter of the network under the community detection process. The partition with the highest value of the modularity score is recorded.

The LP algorithm [19] initializes each node with a unique label. At each iteration of the algorithm, the set of nodes is shuffled. Following the obtained randomized sequence of nodes, each node

adopts a label that most of its neighbors have (if there is more than one label with such property, then one is chosen uniformly randomly). As labels propagate through the network, densely connected sets of nodes form a consensus on their labels. The iterative process of label propagation is repeated until the iteration without any label change. At the end of the algorithm, nodes having the same labels are grouped together as communities. In this study, we performed 100000 runs of the label propagation algorithm for each class collaboration network and recorded the partition with the highest value of the modularity score.

We designed and implemented in Java an algorithm for tracking evolving communities that is able to detect basic evolutionary events related to community evolution: birth of a community, death of a community, merging of two communities, split of a community into two smaller, community growth, community contraction and community stability. For two given partitioned networks N_1 and N_2, the algorithm determines the set of nodes S appearing in both networks (since each node represents a Java class defined in Apache Ant, nodes are matched using fully qualified names of classes). For each two pairs of clusters $c_1 \in N_1$ and $c_2 \in N_2$ the similarity score is calculated by the formula introduced in [20]:

$$sim(c_1, c_2) = min\left(\frac{|c_1' \cap c_2'|}{|c_1'|}, \frac{|c_1' \cap c_2'|}{|c_2'|}\right),$$

where $c_1' = c_1 \cap S$ and $c_2' = c_2 \cap S$.

The best match (BM function) for two pairs of clusters belonging to different networks is the one with the maximal value of the similarity score. Evolutionary events are determined by the following rules:

1. $(\forall i)\ sim(c, c_i) = 0 \iff \text{Death(c)}$
2. $(\forall i)\ sim(c_i, c) = 0 \iff \text{Birth(c)}$
3. $c_1 = BM(c_2) \wedge c_2 = BM(c_1) \wedge |c_1| < |c_2| \iff \text{Growth}(c_1 \rightarrow c_2)$
4. $c_1 = BM(c_2) \wedge c_2 = BM(c_1) \wedge |c_1| > |c_2| \iff \text{Contraction}(c_2 \rightarrow c_1)$
5. $c_1 = BM(c_2) \wedge c_2 = BM(c_1) \wedge |c_1| = |c_2| \iff \text{StableCluster}\{c_1 = c_2\}$
6. $c = BM(c_1) \wedge c = BM(c_2) \Leftrightarrow \text{c} = \text{Split}(c_1, c_2),\ \text{c} \in N_1;\ c_1, c_2 \in N_2$
7. $c = BM(c_1) \wedge c = BM(c_2) \Leftrightarrow \text{c} = \text{Merge}(c_1, c_2),\ \text{c} \in N_2;\ c_1, c_2 \in N_1$

4. RESULTS AND DISCUSSION

The largest weakly connected component of Apache Ant CCN evolves from 77 nodes and 196 links in version 1.1.0 to 821 nodes and 3998 links in version 1.8.2. Each CCN representing one version of Ant has more nodes and links than the CCN representing the previous version. All CCNs are sparse (density is 0.06 for Ant 1.1.0 and drops below 0.02 after Ant 1.4.1). We divided network transitions (change in CCN structure from version v to version $v + 1$) into two categories:

1. Six small transitions (more than one and less than five links added/deleted with the set of nodes remaining the same): 1.5.3 → 1.5.4, 1.6.3 → 1.6.4, 1.4.0 → 1.4.1, 1.6.4 → 1.6.5, 1.5.2 → 1.5.3 and 1.5.0 → 1.5.1
2. Fourteen large transitions (more than 3 nodes added/deleted and more than 5 links added/deleted). The biggest transition is from version 1.4.1 → 1.5.0: 205 new nodes added, 19 nodes deleted, 981 new links added and 177 links deleted.

Since modularity is not scale independent, we normalized modularity values returned by community detection algorithms with $Q_{max}(L) = 1 - 2/\sqrt{L}$, where $Q_{max}(L)$ stands for the maximal value of modularity for a network with L links [6]. Figure 1 shows the evolution of the normalized modularity score for each applied community detection technique. Values of normalized modularity scores vary from 0.35 to 0.47, which is a clear indicator that analyzed CCNs possess community structure by the modularity based definition of community (values above 0.30 are commonly regarded as an indication of significant community structure [6]). It can be seen that in most cases Label Propagation produces community partitions with the highest values of normalized modularity, while the poorest performer is GN, except for the first two versions. Also, from some point in time changes in the normalized modularity score are oscillatory in the range ±0.04, so it can be concluded that the normalized modularity is relatively stable during software evolution.

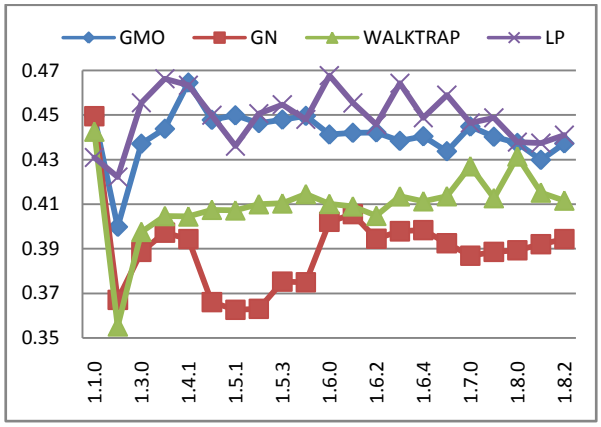

Figure 1. Evolution of the normalized modularity score.

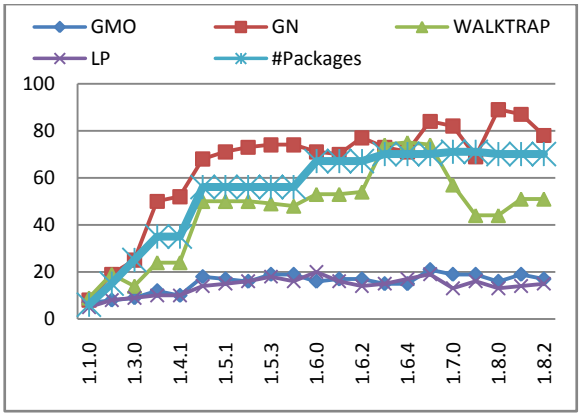

Figure 2. Number of detected communities together with the number of packages (#Packages) existing in Apache Ant.

While different community detection algorithms give relatively close values of the normalized modularity score, the difference between the numbers of detected communities is more drastic. Figure 2 shows the number of detected communities for each clustering technique, together with the number of actual Java packages existing in Apache Ant during software evolution. It can be seen that the numbers of communities produced by LP and GMO are similar and drastically smaller than the numbers of packages and the number of communities produced by Walktrap and GN. LP and GMO give higher values of the normalized modularity score than Walktrap and GM, and the comparison of the number of communities detected using LP and GMO with the

number of packages suggests that the resolution limit problem appeared during community detection for these two techniques.

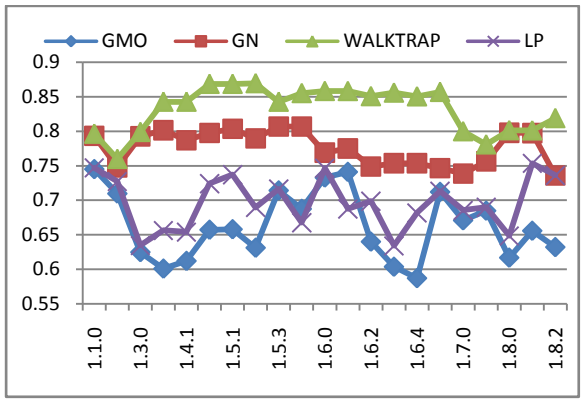

Figure 3. Evolution of the dominant package score.

In order to investigate how obtained community structures correspond to Java package structures, for each version and clustering technique we measured the Dominant Package Score (DPC). DPC for a CCN partition is the average fraction of classes (nodes) belonging to the dominant package in a community, excluding singleton communities (communities that contain exactly one node). Higher values of DPC indicate higher similarity with the package structure. Figure 3 shows the evolution of the dominant package score and it can be clearly seen that Walktrap produces community partitions that are most similar to the package organization of classes defined in Apache Ant.

4.1 Community Structure Quality

Besides the normalized modularity score, we calculated average inter- and intra-cluster density, conductance and expansion for each CCN version and community detection technique, in order to give additional estimates of the quality of obtained community partitions.

Intra-cluster density $D_{in}(C)$ of the sub-network C is defined as the ratio between the number of internal links of C (links joining nodes within C) and the number of all possible internal links. The inter-cluster density $D_{out}(C)$ is the ratio between the number of links running from the nodes in C to the rest of the network and the maximum number of inter-cluster links possible. Density-based definition of a community is related to the comparison of intra-cluster density, overall network density D and inter-cluster density. For C to be a cluster we expect $D_{in}(C) \gg D \gg D_{out}(C)$ [21]. Figure 4 shows the evolution of $D_{in}(C)$, D and $D_{out}(C)$ for Apache Ant. It can be seen that all community detection algorithms produce community structures that satisfy the density based definition of community. Walktrap shows the best performance considering intra- and inter-cluster density: it produces partitions with the highest average values of $D_{in}(C)$ in 17 versions and lowest values of $D_{out}(C)$ in 16 versions.

Conductance $Con(C)$ is the ratio between the number of links that point outside of cluster C and total link volume contained in C. More community-like sub-networks have lower conductance [22]. Expansion $Exp(C)$ also measures the number of links pointing outside C but normalizes by the number of nodes (not the number of all links). Again, lower value of expansion signifies a community. Figure 5 shows the evolution of average conductance and expansion. LP exhibits the best performance by average conductance (21/21 versions), while Walktrap shows the best performance by average expansion (10/21 versions, and we have 7/21 for LP and 3/21 for GMO).

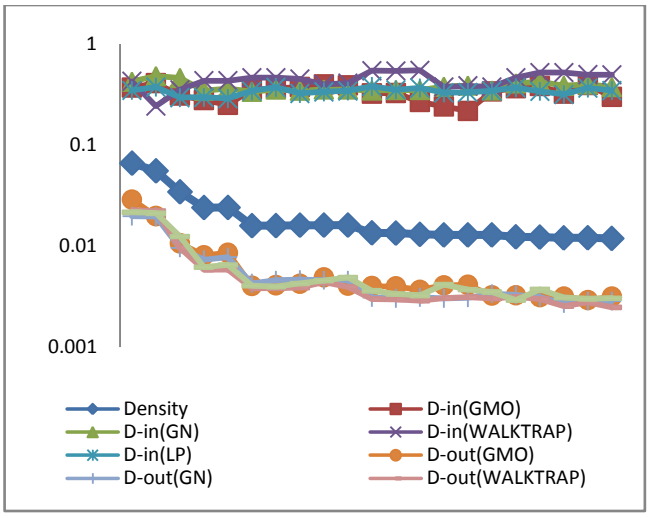

Figure 4. Evolution of average intra-cluster density (D-in), network density (line with rhomboids) and average intra-cluster density (D-out) plotted on semi-log scales.

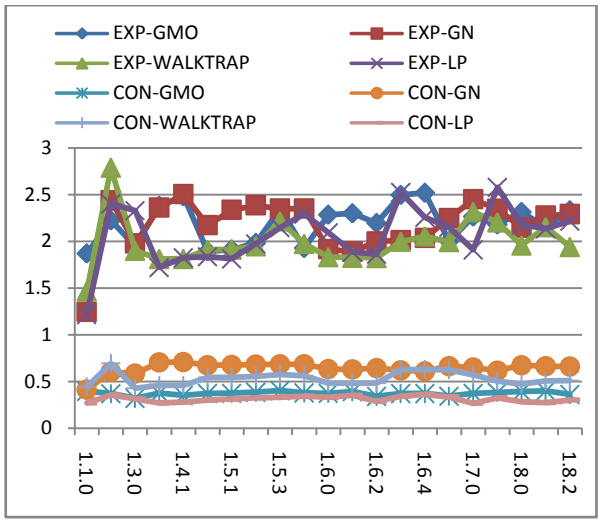

Figure 5. Evolution of average conductance (CON-) and average expansion (EXP-).

4.2 Evolutionary Stability of Community Detection Techniques

In order to investigate evolutionary stability of community detection techniques, we applied an algorithm for tracking evolving communities described in Section 4 on six small network transitions. The basic characteristic of those transitions is that the set of nodes does not change in CCNs representing two successive versions of Apache Ant with the small differences in the set of links.

For each clustering technique, we measured the evolutionary stability factor (ESF), merge factor (MF) and split factor (SF). ESF measures the fraction of nodes in the matched stable, growth and contraction communities. In other words, ESF represents the fraction of nodes that remained in the same cluster after a transition. Stable, growth and contraction clusters are indicators of continuous evolution, while merge and split events indicate dramatic change in network structure on small transitions. Let T denote network transition $N_1 \rightarrow N_2$. MF represents the number of merge events in T (the number of cluster in N_1 that are merged in

N_2) normalized by the number of clusters in N_1. SF measures the number of split events in T (the number of clusters in N_2 that are created by splitting clusters in N_1) normalized by the number of clusters in N_2. If the ESF value is equal to one then we have the same partitioning in N_1 and N_2. On small transitions, higher values of EFS indicate evolutionary stability and evolutionary traceability of communities, while higher values of MF and SF suggest evolutionary degeneracy – extreme sensitivity of the community detection algorithm to small changes in network structure, resulting in drastically different community partitions.

The values of ESF for small transitions are summarized in Table 1, while the values of MF and SF are shown in Table 2. It can be seen that Walktrap and GN have extremely higher values of ESF (always higher than 0.94 and in some case exactly 1) than GMO and LP, whose average ESF scores are 0.74 and 0.65, respectively. On the other hand, Walktrap exhibits significantly smaller values of SF and MF (always smaller than 0.06 and in more than half the cases exactly 0) compared to other algorithms.

Table 1. Evolutionary stability factor on small network transitions

Transition	GMO	GN	WALKTRAP	LP
1.4.0 → 1.4.1	0.54	0.94	1	0.91
1.5.0 → 1.5.1	0.96	0.96	1	0.73
1.5.2 → 1.5.3	0.74	0.96	0.95	0.49
1.5.3 → 1.5.4	0.73	1	0.95	0.64
1.6.3 → 1.6.4	0.96	0.97	0.99	0.56
1.6.4 → 1.6.5	0.51	0.95	0.99	0.6

Table2. Split and merge factor on small network transitions

Transition	GMO		GN	
	SF	MF	SF	MF
1.4.0 → 1.4.1	0.3	0.5	0.08	0.04
1.5.0 → 1.5.1	0	0.06	0.04	0
1.5.2 → 1.5.3	0.26	0.12	0.04	0.03
1.5.3 → 1.5.4	0.15	0.15	0	0
1.6.3 → 1.6.4	0	0	0.01	0.04
1.6.4 → 1.6.5	0.38	0.27	0.17	0.01
	WALKTRAP		LP	
1.4.0 → 1.4.1	0	0	0.1	0.1
1.5.0 → 1.5.1	0	0	0.27	0.21
1.5.2 → 1.5.3	0.04	0.06	0.28	0.19
1.5.3 → 1.5.4	0	0.02	0.19	0.28
1.6.3 → 1.6.4	0.01	0	0.36	0.27
1.6.4 → 1.6.5	0	0.01	0.22	0.12

To summarize, LP shows the best performance considering the normalized modularity score and average conductance, while Walktrap shows the best performance considering the inter- and intra-cluster density, average expansion, dominant package score and evolutionary aspects (ESF, SF and MF scores).

4.3 Coarse-Grained Descriptions

Since Walktrap gives the best performance considering at the same time evolutionary aspects, correspondence with the package structure and cluster quality metrics, we constructed coarse-grained descriptions (CGDs) of CCNs using community structures computed by this method. A CGD is a network of communities: the set of CGD nodes corresponds to the set of communities, while CGD directional link $A \rightarrow B$ denotes that there is a class in community A that references another class in community B. CGD nodes are named by the dominant package in the communities or by class names in the case of one node communities. Figure 6 shows the CGD representing Apache Ant in version 1.3.0. The size of each node indicates the size of the appropriate community. Nodes in the same color form strongly connected components (for each two nodes in a strongly connected component there is a directed path connecting them). It can be seen that four largest communities belong to a strongly connected component. The same situation we observed in CGDs representing other examined versions of Apache Ant: CGD nodes representing largest communities are gathered in a strongly connected component of significant size compared to the number of nodes. Figure 7 shows the relative size of the largest strongly connected components (LSCCs) for each version. It can be seen that the size of LSCC varies in 0.3–0.6 range. The presence of large cyclic dependencies in CGDs implies that CGDs strongly deviate from hierarchical structures.

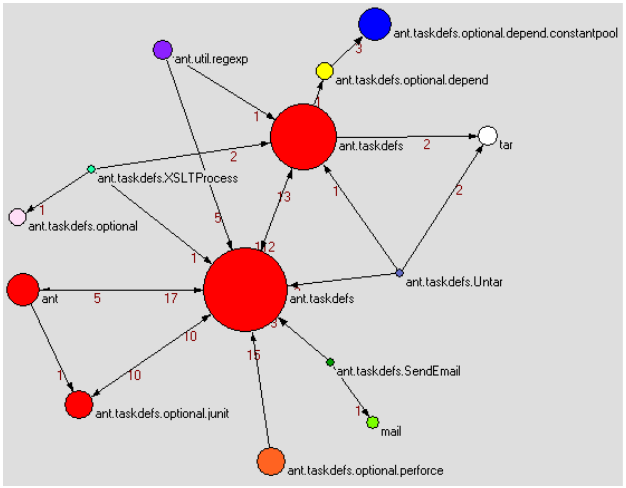

Figure 6. Coarse-grained description of the Apache Ant CCN in version 1.3.0

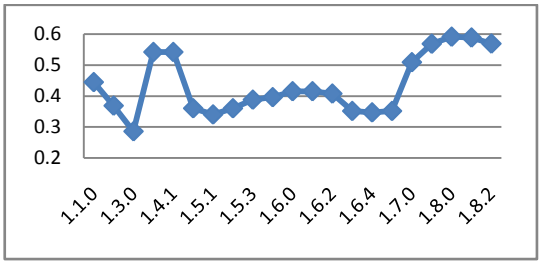

Figure 7. Size of the largest strongly connected component in CGDs.

5. CONCLUSIONS AND FUTURE WORK

This paper investigated four community detection techniques (GN, GMO, Walktrap and LP) applied to class collaboration networks representing the Apache Ant software system in 21 consecutive versions. We showed that the values of the

normalized modularity score are similar and stable during software evolution for each applied technique. However, there are big structural differences in computed partitions. By comparing obtained partitions with package structures, evaluating partitions qualities (using intra- and inter- cluster density, expansion and conductance metrics) and measuring evolutionary stability on small network transitions, we concluded that Walktrap is the best choice for computing communities in the Apache Ant software system. For GMO and LP, we noticed evolutionary degeneracy, a phenomenon where a community detection technique is extremely sensitive to small changes in the class collaboration network. This evidence implies that mentioned techniques cannot be used for clustering because they lead to evolutionary intractable communities. From Walktrap partitions, we constructed coarse-grained descriptions of the Apache Ant class collaboration networks and conducted strongly connected component analysis on them. Results show that CGDs possess large cyclic dependencies, which means that the modular organization cannot be considered as hierarchical.

Community detection methods investigated in this paper produce non-overlapping partitions. It is still an open question how community detection techniques that produce overlapping or fuzzy partitions (such as the k-clique percolation method) behave when applied to software networks.

ACKNOWLEDGMENTS

The authors acknowledge the support of this work by the Serbian Ministry of Education and Science through project "Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support", no. OI174023.

6. REFERENCES

[1] Albert, R., and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys.* 74 (1), 47–97.

[2] Boccaletti, S., Latora, V., Moreno Y., Chavez, M., Hwang, D. 2006. Complex networks: Structure and dynamics. *Physics Reports* 424, 175–308.

[3] Flake, G. W., Lawrence, S., Giles C. L, and Coetzee, F. M. 2002. Self-organization and identification of Web communities. *IEEE Computer* 35 (3), 66–71.

[4] Newman, M. E. J., and Girvan, M. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E.* 69, 026113.

[5] Bollobás, B. 2001. Random Graphs. Cambridge University Press.

[6] Fortunato, S., and Barthélemy, M. 2007. Resolution limit in community detection. In *Proceedings of the National Academy of Sciences* 104(1), 36–41.

[7] Good, B. H., de Montoye, Y.-A., and Clauset A. 2010. The performance of modularity maximization in practical context. *Phys. Rev. E.* 81, 046106.

[8] Šubelj, L., and Bajec, M. 2011. Community structure of complex software systems. *Physica A.* 390(16), 2968–2975.

[9] Myers, C. R. 2003. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Phys. Rev. E* 68 (4), 046116.

[10] Valverde, R., and Solé, V. 2007. Hierarchical small worlds in software architecture. *Dyn. Contin. Discret. Impuls. Syst. Ser. B: Appl. Algorithms* 14 (S6), 305–315.

[11] Hylland-Wood, D., Carrington, D., and Kaplan S. 2006. Scale-free nature of Java software package, class and method collaboration graphs, Tech. Rep. TR-MS1286, MIND Laboratory, University of Maryland, College Park.

[12] Savić, M., Ivanović, M., and Radovanović, M. 2011. Characteristics of class collaboration networks in large Java software projects, *Information Technology and Control* 40 (1), 45–54.

[13] Savić, M., Ivanović, M., and Radovanović, M. 2011. Connectivity properties of the Apache Ant class collaboration network. In *Proceedings of the 15th International Conference on System Theory, Control and Computing*, 544–549.

[14] Dietrich, J., Yakovlev, V., McCartin, C., Jenson, G., and Duchrow, M. 2008. Cluster analysis of Java dependency graphs. In *Proceedings of the 4th ACM Symposium on Software Visualization*, 91–94.

[15] Paymal, P., Patil, R., Bhomwick, S., and Siy, H. 2011. Empirical study of software evolution using community detection. Preprint available at http://cs.unomaha.edu/~bhowmick/STARyNet/papers/techshort.pdf

[16] Csárdi, G., and Nepusz, T. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695. The library can be downloaded at http://igraph.sourceforge.net/

[17] Clauset, A., Newman, M. E. J., and Moore, C. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70, 066116.

[18] Pons, P. and Latapy, M. 2004. Computing communities in large networks using random walks. J. Graph Algorithms Appl. 10 (2), 191–218.

[19] Raghavan, U. N., Albert, R., and Kumara, S. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76, 03616.

[20] Hopcroft, J. E., Khan, O., Kulis, B., and Selman, B. 2004. Tracking evolving communities in large linked networks. *Proc. Natl. Acad. Sci.* 101, 5249–5253.

[21] Fortunato, S. 2010. Community detection in graphs. *Physics Reports* 486, 75–174.

[22] Leskovec, J., Lang, K. J., and Mahoney M. 2010. Empirical comparison of algorithms for network community detection. In Proceedings of the 19th international conference on the World Wide Web, 631–640.