Improving community detection in networks by targeted node removal

Haoran Wen, E. A. Leicht, and Raissa M. D'Souza

Phys. Rev. E **83**, 016114 — Published 28 January 2011

DOI: 10.1103/PhysRevE.83.016114

Improving community detection in networks by targeted node removal

Haoran Wen, E. A. Leicht, and Raissa M. D'Souza†
Department of Mechanical and Aerospace Engineering, University of California, Davis 95616
†*Department of Computer Science, University of California, Davis 95616 and*
†*Santa Fe Institute, 1399 Hyde Park Road, Santa Fe NM 87501**

How a network breaks up into sub-networks or communities is of wide interest. Here we show that vertices connected to many other vertices across a network can disturb the community structures of otherwise ordered networks, introducing noise. We investigate strategies to identify and remove noisy vertices ("violators"), and develop a quantitative approach using statistical breakpoints to identify when the largest enhancement to a modularity measure is achieved. We show that removing nodes thus identified reduces noise in detected community structures for a range of different types of real networks in software systems and in biological systems.

PACS numbers: 89.75.Hc, 89.75.Kd, 87.18.Tt, 05.10.-a

I. INTRODUCTION

The past decade has seen a surge of interest in the network representation of a wide range of real world systems, from social relationships among individuals, to interactions of proteins in biological systems, to the interdependence of function calls in large software projects. Often one wishes to divide a large network up into smaller related subnetworks, called "communities" or "modules". This has been approached using a wide variety of techniques [1–10]. In real world networks, such community structures may represent important groupings identifying common background, interest or function [1].

The notion of overlapping communities, where nodes are allowed to participate simultaneously in more than one subnetwork, was more recently introduced [11–18]. Although such considerations greatly expand the applicability of community structures, there still remain situations where nodes of relatively high degree can connect across the network, disturbing the detected community structures of otherwise well-ordered networks. (See Fig. 1 for an illustration.) Rather than belonging to multiple communities, such "noisy" nodes do not belong preferentially to any community. Here we study several networks present in Open Source Software (OSS) systems and biological systems where this is the case. We introduce methodology for identifying such nodes and develop a quantitative criteria for their removal, showing the improvement in the quality of the community structures that results. We provide a comparison with overlapping community detection methods as well.

Relevant background and related work on community structure and OSS networks is reviewed in Sec. II. In Sec. III we develop techniques to reduce noise in community structure, investigating high degree removal, high modularity removal and the use of statistical breakpoints. Also in Sec. III we apply these techniques to OSS networks and several biological networks. In Sec. IV we compare results obtained with our methods to those resulting from overlapping community algorithms. Overall conclusions are presented in Sec. V.

*Electronic address: {hrwen,eleicht,rmdsouza}@ucdavis.edu

II. BACKGROUND AND RELATED WORK

A. Empirical studies of noise

In some situations it is possible to use expert or domain knowledge to identify *a priori* nodes which may introduce noise, such as in a study by Bird *et al.* analyzing the email communication network of OSS developers [19]. They hypothesized that developers would divide into communities which parallel working groups within the project yet found that a small set of developers (two or three per project) tended to connect to all communities without preference. Such developers were presumed to be project leaders or founders who communicated extensively with others regardless of working group and these vertices were removed manually prior to deploying community detection algorithms. Similarly, a study of *currency* and *commodity* metabolites in modular metabolic networks by Huss and Holme identified the 10 highest degree nodes as "currency metabolites" which they remove manually before running community detection algorithms [20].

In both [19] and [20], high-degree vertices were found to be noisy. Figure 1 illustrates this concept. We generate a small network of 40 vertices where we impose a community structure by initially assigning each vertex to one of four different communities at random. We also assign to each vertex a degree $1 \leq k \leq 30$ drawn from a power-law distribution, $p_k \propto k^{-2.5}$. The two vertices with the largest degree are chosen to be noisy vertices or *violators* which connect without preference to any other vertex in the network. The remaining nodes connect preferentially within their communities, with approximately 85% of their edges falling within their assigned communities. Figure 1(a) shows the communities found in this

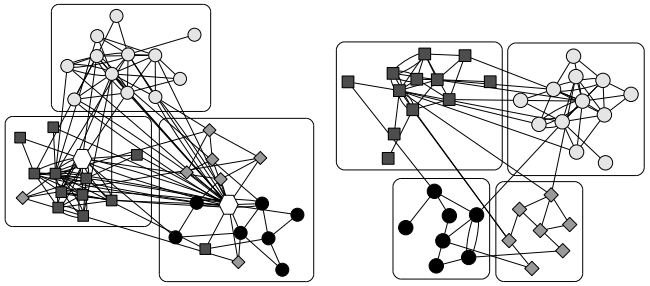

FIG. 1: Community structures of a sample network (a) before and (b) after violator removal, showing the effect of 2 violators out of 40 nodes. Vertices are initially assigned to one of four communities at random (identified via node shape and shading). The two vertices with the largest degree are violators (hexagons) and do not preferentially connect to any single community. The remaining edges fall within communities with 85% probability and between them with 15%. Boxes illustrate the communities detected using the algorithm in [4].

network when using the spectral partitioning algorithm of Newman [4]. The large boxes indicate the detected communities, while colors and shapes represent the communities to which the vertices were initially assigned; the two violators are shown as hexagons. The algorithm finds only three communities, two of which blend the assigned communities. As expected, if we remove the two violators and run the same algorithm we recover the initially assigned community structure as seen in Fig. 1(b). In this small example it was known *a priori* the number and identity of the violators. The methods we develop herein allow detection of vertices acting as violators without needing *a priori* knowledge or requiring that violators necessarily be the largest degree vertices.

B. Community detection algorithms

There are numerous methods available to detect community structure in networks, with comprehensive reviews in Refs. [21–23]. Due to its simplicity and easy extension to directed and weighted networks, we choose as our base algorithm the spectral partitioning algorithm of Newman, which calculates the leading eigenvector of a modularity matrix to identify a network's modular structure [4]. However, our methods should easily generalize to alternate community finding algorithms.

The chosen spectral partitioning algorithm determines the number of communities via an optimization process. It divides a network into communities by maximizing a quality of modularity function Q. Here Q measures the difference between the actual and expected number of edges falling within a community, and is normalized to range from zero to one. Specifically,

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(G_i, G_j), \qquad (1)$$

where A_{ij} is the ij'th element of the adjacency matrix, k_i and k_j are the degrees of vertices i and j respectively, $\delta(r, s)$ is the Krönecker delta, G_i and G_j are the groups vertex i and j are assigned to, and m is the number of edges in the network [4]. The goal of the algorithm is to assign vertices to groups G_i and G_j so that Q in Eq. 1 is maximized, with this maximal Q representing how well a network breaks apart into sub-groups. Q is not a property of a network, instead it is a measure of the quality of a specific grouping of a network [10]. Finding the global optimal assignment is NP hard, so one turns to methods of approximation, some of which are stochastic. As a result, one may not be able to ascertain a unique maximal value of Q for a given network. Q was originally defined for simple networks, but has now been extended to weighted and directed networks [24–26].

Recently, several researchers have developed methods for detecting community structure when vertices may belong to multiple communities and communities may thus overlap [11–16]. However, in a network where some vertices belong to one-quarter or one-half of all communities detected (as we show later is the case for many networks) the question arises as to whether these vertices truly participate in all of the communities. Regardless, such vertices can act as violators obscuring underlying community structures.

C. Empirical OSS Networks

Both software developer communication networks and software dependency networks have attracted much research attention [27–29]. In this work we examine the developer communication networks from four OSS projects: Apache HTTP server, Python, PostgresSQL, and Perl. Each network is constructed by combining monthly mailing list archives of the respective OSS project into a cumulative view spanning several years [30–32]. In these networks, a vertex is a developer and weighted directed edges represent emails between developers. The size of these projects range between 1,000 and 2,500 developers. Developers migrate in out of projects over time [32], making the email social networks noisier than the Apache callgraph network described next. As mentioned, project leaders can participate broadly across the network and introduce noise in community structures.

We also study the code base for the Apache 2.0 HTTP server, an OSS project written in the C programming language (a procedural, rather than Objected-Oriented, software system). Our data is composed of monthly snapshots of the functions and function calls over a four-year period. (For details on the extraction procedure see [33].) The call-graph network is built by considering each function as a vertex and each function call a directed edge. The Apache 2.0's callgraph is extremely stable [34], with data from each snapshot yielding similar results, thus we report on a representative snapshot (10/1/2001).

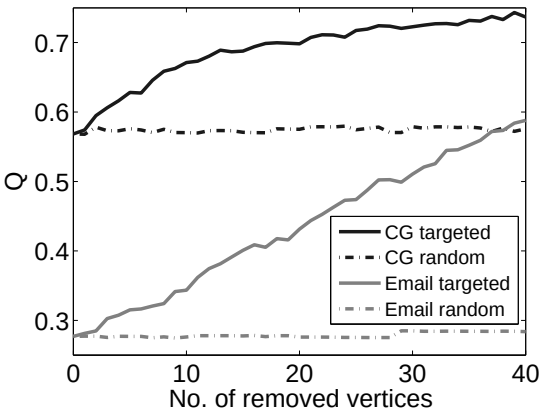

FIG. 2: Change in modularity in response to degree targeted and to random vertex removal for the Apache callgraph (CG), with N=2213 vertices, and E=6455 edges, and the Apache email social network (Email), with N=1232 and E=8064.

III. REMOVING NOISY VERTICES

We hypothesize that, for the software callgraph, low-level functions which are called commonly by other functions across the network disrupt community detection. This is analogous to the empirical observations that project leaders in OSS projects, who have high degree, and currency metabolites, such as H_2O and CO_2, interfere with community detection [19, 20]. The in-degree of a function is the number of other functions it is called by, thus low-level functions have high in-degree. Note, for Apache, the largest in-degrees are an order of magnitude larger than the largest out-degrees. (Ref. [34] contains a discussion of constraints giving rise to this disparity.)

In the remainder of this section we show that degree-targeted node removal is not always satisfactory and that better results are obtained by targeting removal of nodes that cause the largest increase in Q. We also introduce the statistical technique of change point detection as a criteria to determine when to stop node removal.

A. Degree targeted removal

To assess the impact on modularity of removing vertices successively from a network, in Fig. 2 we show the modularity for the Apache callgraph (dark line) and the Apache developer email network (light line) for both degree targeted (solid line) and random removal (dashed line) of vertices. (Under degree targeted removal, we recalculate the degrees of every vertex after each removal.) Note, the modularity is calculated for the largest connected component of the network which remains comparable (more than 80%) with its original size throughout the first 10 removals for both networks and both removal strategies. Under degree targeted removal modularity increases very rapidly. Yet, it continually increases with subsequent vertex removal within the regime shown. Thus Q alone does not indicate the relative value gained by that node removal and hence when to stop removal.

As shown previously, dependent upon the degree sequence it is possible for a network constructed at random to exhibit relatively high values of modularity [35–37]. We consider this the modularity inherent in the degree sequence and denote it by Q_{config}. Thus, we consider the difference between the actual value of Q (as shown in Fig. 2) and Q_{config}, the value for that particular degree sequence, and call $Q - Q_{\text{config}}$ the *absolute modularity* of the real network. A maximum in $Q - Q_{\text{config}}$ after a certain number of vertex removals indicates that the removed vertices where adversely impacting the modularity score more than would be expected by random chance.

In [35] a simple function is introduced as an ansatz to approximate Q_{config}, while in [36, 37] more complex mathematical methods are proposed. We find a simpler approach is sufficient. We take the exact degree sequence of the network of interest and generate an ensemble of 10 random networks with this same degree sequence via the configuration model [38, 39]. Then we calculate the values of inherent modularity, Q_{config}, by averaging over these generated networks. Note that increasing the number of random networks beyond 10 does not change the results for Q_{config}, but only adds computational cost.

In Fig. 3(a) we plot Q for the Apache callgraph and Q_{config} of the corresponding degree sequence as a function of degree-targeted node removal (with their difference $Q - Q_{\text{config}}$ plotted inset). The inset plot shows that $Q - Q_{\text{config}}$ reaches a maximum value after approximately 15 vertices are removed, a reasonable number of low-level functions. However, the data is noisy and several local maxima exist as well. We plot the same quantities for the Apache email network in Fig. 3(b). The inset plot of $Q - Q_{\text{config}}$ is very noisy and reaches its maximal value after more than 40 vertices removed, which is an order of magnitude larger than the two to three developers identified in [19]. We find similar results to those in Fig. 3(b) for the other three email networks studied (Python, PostgresSQL, and Perl). In summary, for the callgraph networks high-degree node removal is plausible, with the location of the peak in $Q - Q_{\text{config}}$ providing a criteria for when to stop removing vertices. The email networks, however, are not amenable to this treatment.

B. High ΔQ targeted removal

Degree-targeted removal provides a starting point, yet the highest degree vertices are not necessarily violators. For instance in the OSS email networks it is possible that a developer with very high degree may be strongly tied to just one community. To develop intuition on methods to remove the correct nodes from a network efficiently, we first simulate a series of networks in which noisy vertices are known, enforcing that the simulated networks

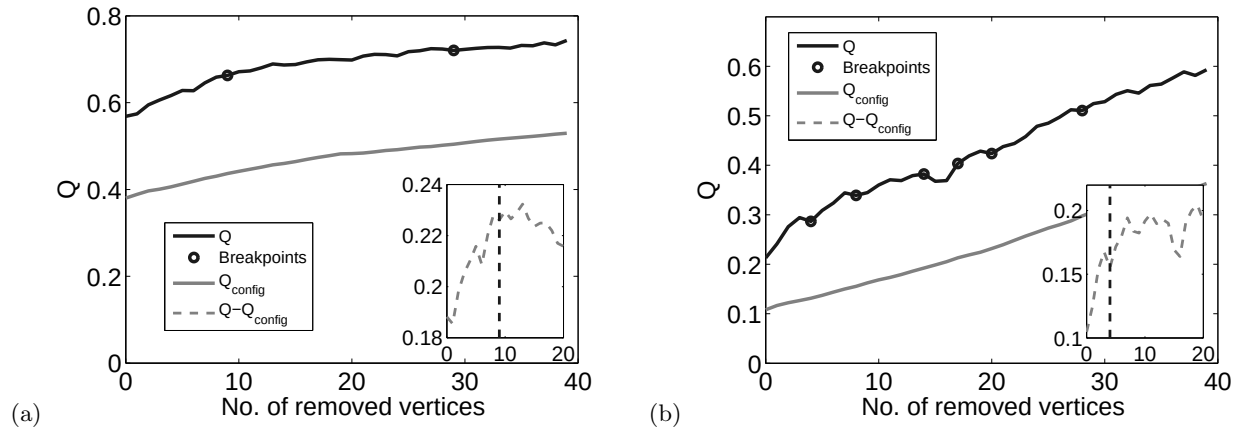

FIG. 3: Q for the real networks plotted in black and the modularity of an ensemble of random networks with a degree sequence identical to the real network, $Q_{\rm config}$, in gray, both as functions of in-degree targeted removal for (a) the Apache callgraph and (b) the Apache developer email network. The points in the main figure indicate "breakpoints" as discussed in Sec. III C. The inset plot shows $Q - Q_{\rm config}$, with the vertical dashed line indicating the location of the first breakpoint.

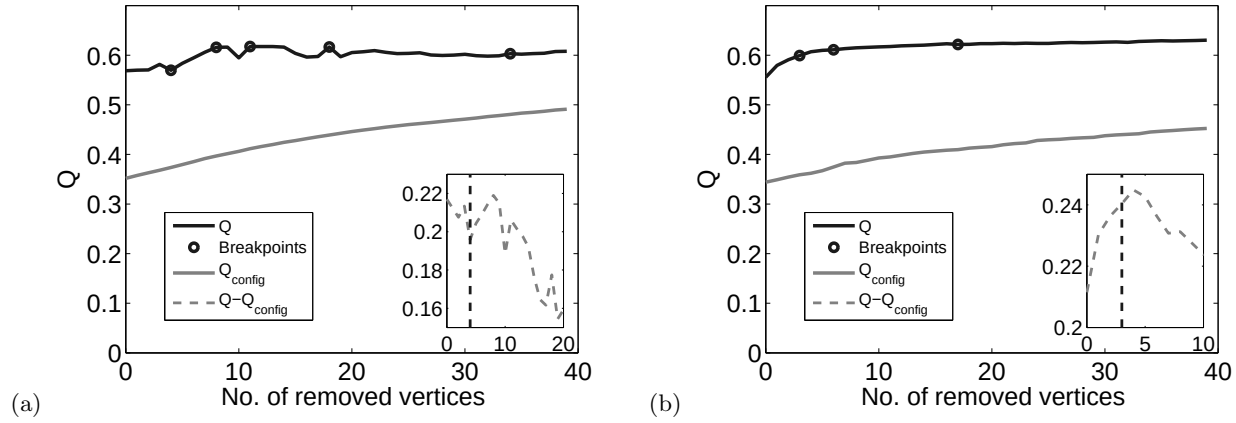

FIG. 4: Modularity change for simulated networks having the degree sequence of the Apache callgraph and with three nodes set to be violators. Inset is $Q - Q_{\rm config}$, with the vertical dashed line indicating the location of the first breakpoint. (a) Modularity change by removing high degree vertices. (b) Modularity change by removing high ΔQ vertices.

have the same degree sequences as the real networks of interest. We construct the model networks by assigning most vertices to one of four predefined communities at random, while three vertices out of the 10 highest degree ones are selected at random to be violators which connect without preference across the network. We use a modified configuration model technique where an edge was allowed to fall between non-violator vertices in different communities with a probability, $p_{\rm between} = 0.15$ and between non-violator vertices in the same community with a probability, $p_{\rm within} = 0.85$. After constructing this model network we identify the communities and calculate the modularity via maximizing Eq. 1.

We exhaustively search through each vertex in the network and identify the vertex who's removal would lead to the largest increase in modularity ΔQ, and remove it. We then apply this technique recursively to the remaining network to find the next noisy vertex, and so on. Notice here ΔQ values must be recalculated after each vertex removal, making this strategy time-consuming. (In practice, for the simulated networks and all the real networks studied herein, ranging to a few thousand nodes in size, testing the 20 highest degree nodes present at any moment is sufficient; increasing the number further does not change the results obtained.)

Results for the simulated networks with the degree sequence of Apache callgraph, generated as described above, are shown in Fig. 4. The modularity change in response to degree targeted removal is shown Fig. 4(a), while response to high ΔQ removal is shown in Fig. 4(b). In Fig. 4(a), we see both Q and $Q - Q_{\rm config}$ are extremely noisy. Furthermore, on examining the identities of vertices, we find that the high degree vertices removed first are not the violators we had intended to remove. As

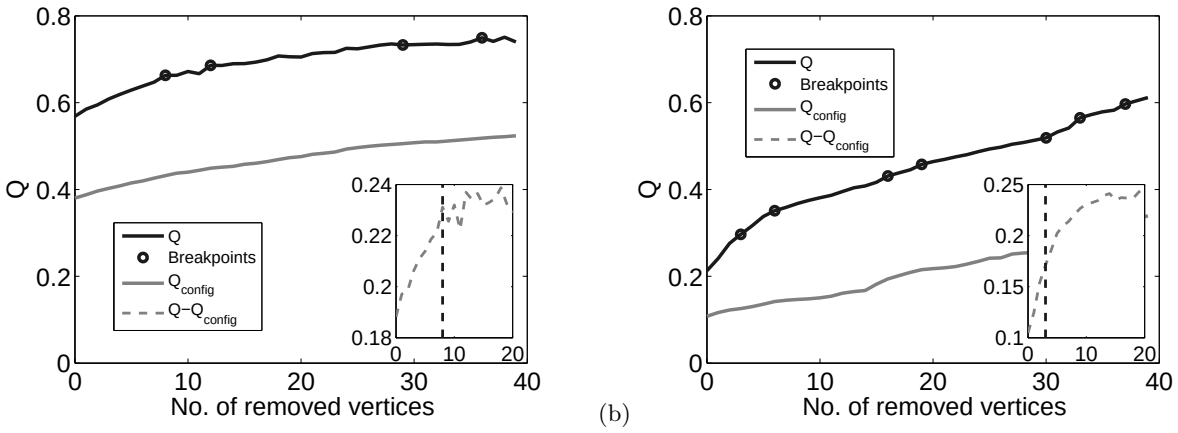

FIG. 5: Modularity change resulting from successively removing high ΔQ vertices, shown for (a) the Apache callgraph and (b) the Apache email network. The inset plot shows $Q - Q_{\text{config}}$, with the vertical dashed line indicating the first breakpoint.

shown in Fig. 4(b), Q increases rapidly as the first three nodes are removed under high ΔQ removal. On inspecting the identity of these three vertices we find that they are exactly the three nodes set initially to be violators.

As seen in Fig. 4(b), $Q - Q_{\text{config}}$ reaches its maximum, not right at three as we would expect, but after three nodes are removed. Thus a peak in $Q - Q_{\text{config}}$ does not provide a rigorous criteria to determine when to stop node removal (as will be developed in Sec. III C). Note, we also examined different number of violators and different simulated networks (e.g. with the degree sequence of developer email network) and all show similar results: modularity increases rapidly while violators are being removed and then stabilizes.

When we apply the high ΔQ removal method to the data for the real Apache callgraph, as shown in Fig. 5 (a), we find similar results to those found based upon removing vertices simply by decreasing degree (shown in Fig. 3(a)). In particular, we find that almost the identical nodes are removed by both techniques.

Results for high ΔQ removal for the real Apache developer email network are shown in Fig. 5(b). Although Q increases much more smoothly than via degree targeted removal (shown in Fig. 3(b)), we still cannot identify a clear global or local maximum in the quantity $Q - Q_{\text{config}}$. In comparing the degree-target removal and high ΔQ removal methods applied to this data set, we find substantial differences in the vertices removed indicating that the highest degree vertices are not necessarily the most noisy. We show only the results for the Apache developer email network, however the email networks for the other projects behave very similarly.

C. Breakpoints to identify the number of violators

To develop a rigorous criteria for determining the number of noisy vertices, we turn to a well-known technique from statistics called change-point detection [40]. This technique was originally used to quantify structural changes in linear regression, identifying *breakpoints* which divide the data into distinct segments where, within each segment, the regression relationship is stable while the coefficients of the regressions vary between the distinct segments. We utilize a dynamic programming algorithm to estimate the optimal breakpoints [41]. Additional details of the algorithm can be found in [40] and [42]. The breakpoints found are indicated in Figs. 3-5. For all these simulated and real data sets, we find that the first breakpoint is the appropriate stopping point for removing vertices acting as violators. Likewise, similar exact agreement between location of breakpoints and the number of violators was found when we implement different numbers of violators in the simulated networks.

In all of our examples we analyze Q as the first 60 nodes are removed (with figures focusing on the first 40). If we look only at $n < 60$ points, the exact number and location of the breakpoints can vary, but the location of the first breakpoint is extremely stable so long as the data sample can be divided into at least three breakpoints.

D. Interpreting results for OSS networks

For the real Apache callgraph data, as shown in Fig. 5 (a), the first breakpoint occurs at the point where 8 vertices have been removed. Four of these 8 functions come from APR, a low-level interface library. The other four are from the C standard library. All these eight functions are basic and low-level with general functionality. Yet, there exist even lower level functions which are not identified as violators. Upon manual inspection we find that these lower level functions have very specific functionality and belong to only one or a few communities.

Manually inspecting the communities detected in the Apache callgraph after violator removal, we find that

each community roughly corresponds to a group of functions which complete one relatively independent or several similar tasks. These communities typically contain only functions belonging to one or a few distinct modules, where modules are organized software sub-projects with different concerns. For example, there is an identified community consisting only of functions from a single module, called SDBM (which handles database transactions), and the low-level functions they call. These coherent communities, which reflect the function of the project, were originally hidden and only emerge as we iterate the removal of violators.

Under high ΔQ removal three of the OSS developer email networks (Apache, Perl and Python) have breakpoints when three vertices are removed, while for Postgres the breakpoint occurs when four vertices are removed. The four OSS projects have different management styles. Apache is a foundation based project with a group of core developers and several more minor developers. Perl and Python are considered Monarchies, with only a few core developers. Postgres is a community without identified project leaders. It is reasonable that in project with only a few core developers one finds violators. More interesting is that even in a project structured like Postgres (with dynamic self-organized communities), there can exist a small number of "noisy" developers.

Commensurate with the intuition put forth by Bird *et al.* we find approximately three violators in each of the developer email networks [19]. Bird *et al.* used an *a priori* criteria of removing the three vertices with the highest betweenness centrality. In both Apache and Perl, two of the three violators identified by our technique are also among the three highest betweenness vertices; in Postgres, three of the four violators are the top three betweenness vertices; but in Python, only one of the three violators identified are among the top three betweenness vertices. Similar to how degree targeted removal can result in removing high-degree nodes which are not violators, our results indicate that previous methods may have wrongly removed vertices with very high betweenness that where not violating community structures.

Note that removing vertices necessarily reduces the size of the largest connected component in a network, but not significantly. For example, with all 8 violators removed, the largest connected component of the Apache callgraph network is 97% of its original size. For all four email social networks after violator removal, the largest connected component is at least 80% of its original size.

E. Application to biological networks

We apply our technique to different types of networks present in biological systems. In particular, we investigate two gene regulation networks (*E. coli* and yeast transcription network [43]), one metabolite network (*E. coli* metabolite network [18]), and five protein protein interaction (PPI) networks (all from the online BioGRID database [44]). The two transcription networks are directed and weighted, while the *E. coli* metabolite network and the PPI networks are undirected and unweighted.

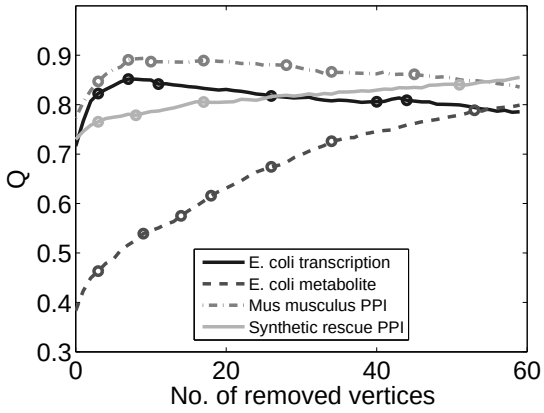

FIG. 6: Modularity change for four biological networks with high ΔQ vertex removal. Breakpoints for each network are shown respectively; all the first breakpoints occur at three.

In the yeast transcription network and three of the PPI networks, we find that removing vertices does not increase the modularity. Hence, we find no vertices acting as violators in these networks. In the *E. coli* transcription network and *E. coli* metabolite network, noisy vertices are identified. As shown in Fig. 6 there are three violators identified at the first breakpoint for *E. coli* transcription network. (In contrast, the difference between Q and Q_{config}, not shown in the figure, reaches its maximum around five.) Likewise, in *E. coli* metabolite network, three violators are identified at the first breakpoint as shown in Fig. 6. These three violators are proton, water and ATP, which were found to belong to high numbers of communities in [18] when overlapping communities were considered (discussed in detail in the next section). For two of the PPI networks from BioGRID (Mus musculus and Synthetic rescue), we find violators by applying our technique, as also shown in Fig. 6.

F. Quantifying the differences before and after violator removal

As discussed above in the context of the Apache callgraph network, using domain knowledge we are able to verify the improvement in community detection when violators are removed. To quantitatively compare the difference in community structure before and after removing violators, we adopt the measure of variation of information (VI) [45]. This normalized measure, ranging between zero and one, is used to compare statistical overlap of two grouping results. When two groupings are exactly the same, the measure is zero. Notice that it compares different groupings of one specific network, thus we only consider vertices present both before and after violator

	Apache callgraph	Apache	Perl	Postgres	Python	*E. coli* trans.	*E. coli* meta.	Mus mus.	Syn. Rescue
VI	0.3222	0.2252	0.1969	0.2390	0.2360	0.0838	0.2362	0.1453	0.2652
Groups	18→ 21(8)	5→6(3)	6→7(3)	7→7(4)	8→7(3)	20→ 18(3)	10→12(3)	40→30(3)	27→34(3)

TABLE I: Comparison of community structures before and after violators removal. The first row is the variation of information between the grouping of the original network and the network with violators removed. The second row shows the change in the number of communities detected, with the number of violators removed shown in parentheses.

removal. Table I shows results for the networks in which violators are detected. Although most of VI values are small, they are significant. (The only exception is *E. coli* transcription network.) The number of communities detected before and after violator removal are shown in the second row of Tab. I, with the number of violators detected indicated in parentheses.

IV. OVERLAPPING VERSUS NOISE IN COMMUNITIES

To understand the role violator vertices play when communities are allowed to overlap we implement two methods for detecting overlapping community structure. The first is the method of Palla *et al.* , which is one of the original such techniques [11, 46]. The method initially detects fully connected subgraphs of size k (*i.e.* k-cliques) and forms communities by grouping together k-cliques which share one or more vertices. Varying k results in detection of different communities, and the authors suggest a typically good choice is $k = 4$ [11]. When the algorithm was applied to our Apache callgraph data only 141 out of the total 2,213 vertices could be assigned to any community (as the algorithm does not guarantee that every vertex will be assigned to at least one community). Yet all eight of the vertices we detected as violators were assigned to one or more communities. Details are in Tab. II, showing the total number of communities detected and the number of communities to which each of the first four violators (out of eight) belong. Of the 141 Apache callgraph vertices assigned to communities, 119 belong to only one community, 12 vertices belong to two communities, and only 10 vertices belong to more than two. As shown in Tab. II the first four violators belong to at least four communities. In particular note apr_pstrdup and strlen which belong to seven and 11 communities respectively, which are over 20% of all communities.

Shen *et al.* proposed a revised version of the Palla *et al.* algorithm [14] which treats the maximal cliques as vertices, and, moreover, guarantees all vertices are assigned to at least one community. Table II shows a comparison of the two methods. The Shen algorithm finds in total fewer communities, yet that the violators belong to multiple. Note, in particular the function *strlen* which is assigned to eight of 18 communities (over 44% of all communities), and the two functions assigned to 4 communities (over 22% of all communities).

Table III summarizes the results when applying these two methods to the other networks. The first five networks listed were found by our high ΔQ method to have disruptive vertices, whereas the final three networks did not show this. The column titled "tot." gives the total number of communities detected via the Palla *et al.* and Shen *et al.* algorithms. The last three columns indicate the number of communities to which the three most noisy vertices identified belong.

When applied to the Apache social network and the *E. coli* network the Palla algorithm does not converge. (The algorithm was run for more than three days without outputting a solution.) Yet the Shen algorithm converges

	tot.	apr_pstrdup	strlen	apr_palloc	apr_pstrcat
Palla	31	7	11	4	6
Shen	18	4	8	2	4

TABLE II: The total number of communities detected (column 2) for the Apache callgraph data using two different algorithms, labeled Palla [11] and Shen [14] respectively. The remaining columns show the identity of the first four violators and the number of communities each violator is assigned to by each algorithm. (The clique size used is $k = 4$.)

	algm.	tot.	v. 1	v. 2	v. 3
Apache social	Shen	35	11	12	9
E. coli trans.	Shen	12	1	1	1
E. coli metabolite	Palla	15	4	3	1
	Shen	9	9	9	9
Mus musculus	Palla	3	0	0	0
	Shen	23	1	1	1
Synthetic Rescue	Palla	8	1	0	1
	Shen	25	1	1	1
Yeast	Palla	1	0	0	0
	Shen	17	1	3	1
Co-purification	Palla	28	1	1	1
	Shen	15	1	1	1
Dosage Rescue	Palla	28	3	1	0
	Shen	31	1	1	1

TABLE III: Total number of communities identified by the two algorithms [11] and [14], and the total number of communities to which each of the first three vertices removed are assigned. The first five networks listed were found via our high ΔQ removal technique to have disruptive vertices, while no such vertices are found in the final three networks.

for all of the networks. As seen in Tab. III, in most cases the violators identified by our technique are those vertices which belong to many communities. Of particular note is the *E. coli* metabolite network where the three violators (proton, water and ATP) are found to belong to every community identified by the Shen algorithm. It is also noteworthy that the Mus musculus and the Synthetic Rescue networks show no indication of being noisy when viewed from the overlapping community lens, yet do show noise when treated by our techniques of high ΔQ removal and breakpoints. Our technique, therefore, can identify noise in scenarios where overlapping community algorithms would not provide an indication.

V. CONCLUSIONS

Significant noise in the community structures of networks, even those comprised of thousands of nodes, can be introduced by just a few nodes. This phenomena was identified previously using *a priori* knowledge [19] and heuristic techniques such as removal of the top ten highest degree nodes [20]. In the work presented here we develop a quantitative framework for identifying and removing noisy nodes. While high degree vertices tend to be more noisy than low degree vertices, we show in several places that the most noisy vertex is not necessarily the vertex with highest degree. We instead develop a procedure to identify and remove noisy nodes based on high ΔQ removal iterated until the first breakpoint in the resulting value of Q. The violators thus identified in email social networks are commensurate with the violators identified using a heuristic approach developed using domain knowledge in [19]. We also show our technique identifies violators successfully in simulated networks and in a series of biological networks. Our technique provides a systematic solution to the problem of identifying noisy vertices and can be especially useful in the absence of domain knowledge.

We also compare our results with results obtained by overlapping community finding algorithms. In many cases the violators identified by our technique belong to an extreme number of communities and should be classified as noise. Yet there are instances where overlapping communities would not indicate evidence of noise yet our technique nonetheless does identify noisy vertices. Our method can be considered orthogonal or complementary to overlapping community finding algorithms.

Here, we use Newman's proceedure [4] as our base algorithm. Any other algorithm which assigns each vertex to a community could be used instead, as our method depends only on calculating the modularity Q for the resulting assignment of vertices to communities. Removing the noisy vertices thus identified can help improve the quality of community structure detected in networks.

VI. ACKNOWLEDGMENTS

We thank C. Bird, M. F. Rahman, P. Devanbu and V. Filkov for useful discussions; A. Dandekar, F. Martinelli, and R. Reagan for Citrus PPI network data; Y.-Y. Ahn and S. Lehmann for the *E. coli* metabolite network data; and H.-W. Shen for providing his executable code. This research was supported in part by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 and the National Science Foundation under Grant No. IIS-0613949.

[1] M. Girvan and M. E. J. Newman. *Proc. Natl. Acad. Sci USA*, 99:7821, 2002.
[2] M. E. J. Newman and M. Girvan. *Phys. Rev. E*, 69:026113, 2004.
[3] M. E. J. Newman. *Phys. Rev. E*, 69:066133, 2004.
[4] M. E. J. Newman. *Phys. Rev. E*, 74:036104, 2006.
[5] M. E. J. Newman. *Proc. Natl. Acad. Sci.*, 103:8577–8582, 2006.
[6] Y. Sun, B. Danila, K. Josic, and K. E. Bassler. *Europhysics Letters*, 86:28004, 2009.
[7] L. Donetti and M. A. Munoz. *Journal of Statistical Mechanics*, 10:10012, 2004.
[8] L. Donetti and M. A. Munoz. In *Modeling cooperative behavior in the social sciences*, volume 779, pages 104–107, 2005.
[9] J. Duch and A. Arenas. *Phys. Rev. E*, 72:027104, 2005.
[10] A. Clauset, M. E. J. Newman, and C. Moore. *Phys. Rev. E*, 70:066111, 2004.
[11] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. *Nature*, 435:814–818, 2005.
[12] S. Gregory. In *Proc. 11th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 91–102, 2007.
[13] N. Du, B. Wang, and B. Wu. In *Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1371–1372, 2008.
[14] H.-W. Shen, X. Cheng, K. Cai, and M.-B. Hu. *Physica A*, 388:1706–1712, 2009.
[15] D. Fiumicello, A. Longheu, and G. Mangioni. Studies in Computational Intelligence. Springer Berlin, 2009.
[16] H.-W. Shen, X.-Q. Cheng, and J.-F. Guo. *arXiv:0905.2666*, 2009.
[17] T. S. Evans and R. Lambiotte. *Phys. Rev. E*, 80:016105, 2009.
[18] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. *Nature*, 466:761–764, 2010.
[19] C. Bird, D. Pattison, R. M. D'Souza, V. Filkov, and P. T. Devanbu. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35, 2008.
[20] M. Huss and P. Holme. *IET Systems Biology*, 1:280–285, 2007.
[21] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. *J. Stat. Mech*, P09008, 2005.

[22] M. A. Porter, J.-P. Onnela, and P. J. Mucha. *Notices of the American Mathematical Society*, 56:1082–1097, 1164–1166, 2009.

[23] S. Fortunato. *Physics Reports*, 486:75C174, 2010.

[24] M. E. J. Newman. *Phys. Rev. E*, 70:056131, 2004.

[25] E. A. Leicht and M. E. J. Newman. *Phys. Rev. Lett*, 100:118703, 2008.

[26] A. Arenas, J. Duch, A. Fernández, and S. Gómez. *New J. Phys.*, 9:176, 2007.

[27] C. R. Myers. *Phys. Rev. E*, 68:046116, 2003.

[28] S. Valverde and R. V. Solé. *Phys. Rev. E*, 76:046118, 2007.

[29] S. Valverde and R. V. Solé. *Dynamics of Continuous Discrete and Impulsive Systems: Series B; Applications and Algorithms*, 14:1–11, 2007.

[30] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143, 2006.

[31] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 185–186, 2006.

[32] C. Bird, A. Gourley, P. T. Devanbu, A. Swaminathan, and G. Hsu. In *Fourth International Workshop on Mining Software Repositories, ICSE Workshops* 2007.

[33] Z. M. Saul, V. Filkov, P. T. Devanbu, and C. Bird. In *Proceedings of the 6th ESEC/SIGSOFT Foundations of Software Engineering*, pages 15–24, 2007.

[34] H. Wen, R. M. D'Souza, Z. M. Saul, and V. Filkov. in *Dynamics On and Of Complex Networks*, pages 199–215. Birkhauser, Springer, 2009.

[35] R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. *Phys. Rev. E*, 70:025101, 2004.

[36] J. Reichardt and S. Bornholdt. *Phys. Rev. E*, 74:016110, 2006.

[37] J. Reichardt and S. Bornholdt. *Phys. Rev. E*, 76:015102, 2007.

[38] B. Bollobás. *European Journal of Combinatorics*, 1:311, 1980.

[39] M. Molloy and B. Reed. *Random Structures and Algorithms*, 6:161–180, 1995.

[40] J. Bai and P. Perron. *J. Appl. Econometrics*, 18:1–22, 2003.

[41] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[42] A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. *J. Statist. Software*, 7:1–38, 2002.

[43] U. Alon. `http://www.weizmann.ac.il/mcb/UriAlon/groupNetworksData.html`.

[44] `http://www.thebiogrid.org`.

[45] B. Karrer, E. Levina, and M. E. J. Newman. *Phys. Rev. E*, 77:046119, 2008.

[46] G. Palla et. al. `http://cfinder.org/`.