DCS: Divide and Conquer Strategy for Detecting Overlapping Communities in Social Graphs

Syed Agha Muhammad
Embedded Sensing Systems
Technische Universität Darmstadt, Germany
muhammad@rbg.informatik.tu-darmstadt.de

Kristof Van Laerhoven
Embedded Systems
Universität Freiburg, Germany
kristof@ese.uni-freiburg.de

ABSTRACT

Detecting meaningful communities in real-world networks such as large social graphs is a problem of considerable interest, as witnessed by a large amount of literature in this field. In this work, we focus on two important and often neglected aspects of community detection (CD), namely scalability of approaches on large graphs and algorithm selection for CD. A popular class of community detection methods, such as Louvain method, Infomap and BigClam, works reasonably well on networks with a few thousand nodes. However, when the network size increases, the performance of these algorithms rapidly degrades regarding runtime efficiency and/or the quality of delivered results. In this work, we propose a simple bridge based overlapping community detection algorithm that scales to large networks of millions of nodes and edges. We develop a simple bridging technique to identify functional modules of the graph and then analyze each module independently based on its structure. Further, we propose a simple technique to detect local leaders in each module, and then expand these leader sets using a scoring procedure based on conductance and internal density; finally, the local communities are merged to form a global view of each module. As with any machine learning problem there is no 'one-size-fits-all', similarly, each CD approach focuses on a certain scope of structure and it only excels in a specific part of the problem space. Next, we propose our solution to automatically select a CD algorithm based on the observed structure of a graph. We evaluate our approach against the state-of-the-art CD algorithms and find that our method clearly performs better than others by producing a more accurate estimate of a derived community structure to the ground truth data.

ACM Classification Keywords
I.5.3 Clustering: Algorithms; E.1 Data Structures: Graphs and networks

Author Keywords
bridge nodes, seed and expansion, algorithm selection, overlapping communities

INTRODUCTION

Social network analysis has emerged as one of the most exciting domains of data analysis and data mining over the last decade. The scientific collaboration, the World Wide Web, business organizations and biological networks are common examples of complex networks. One of its most interesting subfields is community detection (CD). A community corresponds to a group of nodes with denser connections amongst its members than between its members and the remainder of the network. Similarly, community detection, loosely speaking, is any process that takes a graph as an input and produces sets of related nodes or so called 'communities'. Detecting communities in social graphs or protein networks is a problem of considerable interest that has received enormous attention [1, 4, 7, 8, 23]. Social graphs are normally huge, it is often challenging to design CD algorithms that produce strong results on all parts of the graph. The problem is further complicated by the fact that social graphs have heterogeneous densities, i.e., they display several layers of grouping of the nodes, with small clusters included within large clusters, which are in turn included in larger clusters, and so on. Generally, in the vast literature on CD, the approaches for CD develop an objective function that captures the intuition of a community, and then apply a heuristic to extract a set of nodes that represents the notion of a good community from a global perspective. To keep this manuscript simple, we can say that the generic community detection methods suffer from three major problems. First, a majority of these approaches generally fail on large social networks due to the heterogeneity of the social networks on local and global scales, in such cases, the generic community detection algorithms produce communities that are either too large and incoherent in dense regions or very small in sparse regions. Second, most CD methods have trouble scaling to large networks and further the lack of reliable ground truth makes evaluation surprisingly difficult. Third, each CD approach focuses on a certain scope of structure and it only excels in a specific part of the problem space. In a majority of the cases, when CD algorithms fail to detect the required scope in a graph, they produce outliers.

Present work. In this work, we propose two novel solutions to these community detection problems. Essentially, we adopt a modular approach to detect effective functional modules without causing substantial loss of network structure and integrity. To do so, we develop an efficient local bridging technique to identify 'bridge nodes' that play critical roles between different modules of the graph. Figure 1 illustrates our approach. In practice, we extract the local neighborhood network of each node and then search for the mutual neigh-

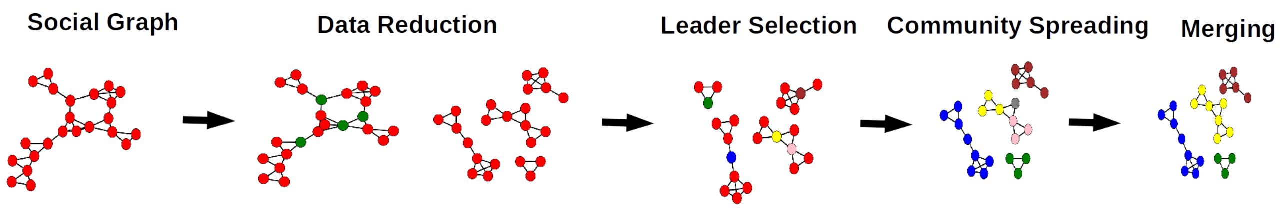


Figure 1. Overview of DCS. We identify bridging nodes (green nodes) and then remove the loosely connected edges. There are no green nodes in the second sub-graph in the 'Data Reduction' phase, because now they are part of the modules. We identify sets of leaders in each submodule and then expand them using a defined scoring function (gray node in the 'Community Spreading' phase represents overlap). The different colors in the 'Leader Selection and Community Spreading' phases represent different leaders and communities that are expanded around them. We merge local communities in each module to get a global perspective.

bors between a pair of nodes that are connected to it, ignoring the presence of the node itself. For each node, we combine the information for the mutual neighbors between its neighbors. From this step, we get a local view around each node that explains how many distinct modules this node belongs to. Further, we leave each node to remain connected to the neighbors with which it shares many mutual neighbors, i.e. where neighbors of a node are also tightly connected to each other, and remove its edges with the remaining loosely connected nodes. From this phase, we obtain a set of modules that are further analyzed to detect communities. For this reason, we name our solution *DCS* (Divide and Conquer Strategy). We also name our solution a modular solution, because we identify modules of the graph and then detect communities in each module. *Why is detecting modules a good idea?* It is a widely accepted observation that complex systems exhibit hierarchical organizations, in which each module represents a bigger picture and it further contains a set of nested communities. Additionally, Balcan et al.[3] have shown empirically and theoretically that hierarchical communities can simultaneously and quantitatively reproduce many commonly observed topological properties of networks. One shall note that we primarily extract the top level hierarchy of the graph. Further, it is computationally fast and (memory) efficient to analyze modules from a large graph. In the second phase, we propose a simple approach to find a set of local leaders from each module. The selected leaders further expand communities around themselves. In the expansion phase, we expand communities around the leading nodes using a defined scoring function that considers local conductance and density to build the communities. Finally, for each module, we merge its local communities to give a global view. *How does DCS handle local heterogeneity?* While merging local communities, we set a different overlap percentage for small, medium and large modules. This step avoids producing large or small communities from all parts of the graph. Last, we propose our solution to automatically select a community detection algorithm based on the observed structure of the modules. We train a well-known supervised classifier on a large set of networks. For the training phase, we study different properties of each graph, and then apply different community detection algorithms on each graph. The algorithm with the highest evaluation score is assigned to a class. In the prediction phase, we extract a set of features from each module and allow the trained classifier to predict the algorithm class. *Why is using multiple CD algorithms on different modules a good idea?* As with any machine learning problem there is no 'one-size-fits-all' and each CD algorithm excels in a certain problem space. By training a complete system of networks having different topological properties, we explore the specific scope of each approach, which can potentially lead to improved results. In summary, the contributions of this work are:

- We introduce a simple, intuitive and cost efficient technique based on local neighborhood information to identify critical nodes (and modules) of the graph.
- We propose an efficient leader selection and community scoring function that delivers low conductance and high internal density communities.
- Experiments reveal that our technique is scalable and produces accurate results. Moreover, we find that *DCS* performs much better than other state-of-the-art methods on large and small networks.
- We develop a simple mechanism of automatic algorithm selection for social graphs.

The rest of the work is organized as follows: Section 2 discusses the related work. Section 3 is dedicated to the problem representation and definition. Section 4 describes the different phases of *DCS*. Section 5 presents the experimental setup. Our experiments are presented in Section 6. We conclude this work in Section 7.

RELATED WORK

Detecting communities in social graphs is a well studied problem in graph mining. A comprehensive survey by Fortunato et al.[10] and Xie et al.[25] explores almost all popular techniques to detect communities. The problem has been addressed using various techniques such as bridge detection [12], modularity optimization [4, 7], information theoretic [23], ego network [8, 22], graph pattern searching [1], non-negative matrix factorization (NMF) [26] and seed based expansion [24]. We discuss only a few of them; especially the ones relevant to this work.

Non-Overlapping CD Algorithms. The bridge based methods [12, 13] separate the communities by identifying the sparser areas of the graph. However, they are slow and only applicable to the sparser graphs with up to 10^4 nodes. The bridge detection algorithms rely on some centrality measures [11], mainly betweenness centrality, to find the most relevant edges or nodes of the graph. However, the fastest betweenness centrality computation time for a graph is bound by $O(nm)$ (n and m denote the number of nodes and edges) on unweighted and $O(nm + n^2 log\ n)$ on weighted graphs, which makes them less effective for huge graphs. A variation of betweenness is proposed to detect the bridging nodes (named as bridge centrality) [14], but it uses betweenness centrality in the core and additionally it is not scalable for large networks. A variety of community detection methods are based on the modularity concept, a quality function of the partition by Newman et al.[7]. Modularity measures internal connections, but it does so with a reference to a randomized null model. Modularity has been very influential in recent community detection studies, the Louvain method [4] is an example of a modularity technique. However, modularity techniques suffer from a well-known resolution limit problem, i.e., bias towards bigger communities. Similarly, Infomap [23] is another popular CD algorithm that is considered to be one of the most reliable non-overlapping techniques for delivering accurate community structures.

Overlapping CD Algorithms. So far, we discussed non-overlapping algorithms, but an important property of community detection is to recover overlapping communities from graphs. In this regard, very relevant to our work are [8, 26]. In their inspiring work, Coscia et al. [8] have proposed a local-first approach for community detection. Their approach forms local communities around each node using label propagation algorithm [21], and later on local communities are merged into a global collection. Similarly, BigClam [26] studies networks as bipartite affiliation model, i.e. nodes of a social network are affiliated with the communities they belong to and the links to the underlying social network are then derived based on the node community affiliation. $NISE$ [24] proposes two methods for choosing the seed set, and then it uses personalized PageRank scheme [2] to expand the seeds until an optimal solution is found.

BACKGROUND

We provide a brief overview of the basic notions that are relevant for this work. We also discuss the Merge function that is frequently used in our approach.

Graph Representation

Let $G = (V, E, W)$ be an undirected, weighted graph, where V is a set of nodes, E is a set of edges and the function $W : E \to \mathbb{R}$ assigns a real value $W(e)$ to each edge $e \subset E$. We interchangeably use the terms vertex and node to refer to the elements of vertex set V, and similarly edge and link connection to refer to the elements of edge set E. The density of a graph G is defined as:

$$D_G = \frac{2 \times |E|}{|V| \times |V - 1|} \tag{1}$$

A dense graph has a number of edges close to the maximal number of edges, further, a graph with only a few edges is a sparse graph. The degree of a node denotes the number of its neighbors. A neighborhood $\Gamma(v)$ of a node is a set of nodes that are directly connected to v via an edge and v itself. The extended degree k_v^+ of a vertex v is the number of edges attached to it plus the number of edges attached to each of its neighbors. The extended degree is defined as:

$$k_v^+ = k_v + \sum_{n \in \Gamma(v)} k_n \tag{2}$$

One type of method examines the graph structure around individual nodes, so-called *ego graphs*. Ego networks are subnetworks that are centered on a certain node. An ego graph $EN(v, G)$ is a sub-graph $\hat{G}(\hat{V}, \hat{E})$ if $\hat{V} \subset V$ and $\hat{E} \subset E$, where $\hat{V}$ represents the direct neighbors of v, and $\hat{E}$ is a subset of E containing all the edges (u, v) where $u \in \hat{V} \wedge v \in \hat{V}$. Further, removing node v and all the edges directly associated to it from a copy of G results in graph-vertex difference $-g : -g(v, G)$. We will use the term $EgoWithoutEgo$ (EWE) for this operation:
$EWE(v, G) = -g(v, EN(v, G))$. These are the two concepts we share with [8]. Basically, the EWE operation results in producing a neighborhood graph around each node. Next, a node is a bridge-node in G if it is located between and connects modules in a graph. In other words, a bridge node, if deleted, would break the graph into two or more modules. Note that our approach can also handle unweighted and directed graphs.

Last, the qualitative definition of a so called 'community' is still ambiguous for complex networks, as there exists none that is universally accepted. However, the widely accepted notion defines a community as a group of nodes having dense connections in between, and sparse connections to the rest of the graph. We use this notion as a foundation for our approach. Basically, we identify big modules of the graph that are connected by weak links to each other, and then further identify communities in each module. Below, we define other intuitive properties for our definition of community:

- Communities are mainly local structures, they involve nodes that are directly connected to each other plus at most an extended neighborhood of them.
- Communities can overlap, as individual nodes may belong to multiple communities.

Property 1 sets an upper bound for the size of communities, otherwise the size of communities will be huge. There are many studies that have used similar assumption in their work [1, 8, 16]. In fact, [1] have discussed that how the huge communities delivered by CD algorithms are meaninglessness and incoherent. Last, this property is also in agreement with the sociological studies; especially with social balance theory and Triadic closure. Easley et al. [9] states in their work, " If two people in a social network have a friend in common, then there is an increased likelihood that they will become friends themselves at some point in the future". Considering this, our assumption is certainly logical.

Merge Function

The pseudo code of the Merge function is specified in Algorithm 1. We slightly modify the merging function proposed in [8]; however, the main concepts are inherited from their work. While the former only merge communities greater than a specified length ($|L|$), we remove that condition for our approach. The value of ϵ triggers the percentage of overlap between the lists. Two lists A and L are merged if and only if at least $\epsilon\%$ of one list is included in the other one; in this case, L and A are removed from list set $\mathcal{L}$ and their union is added to the resulting set. The value $\epsilon = 0$ would even merge lists that do not share a single common element, on the other side

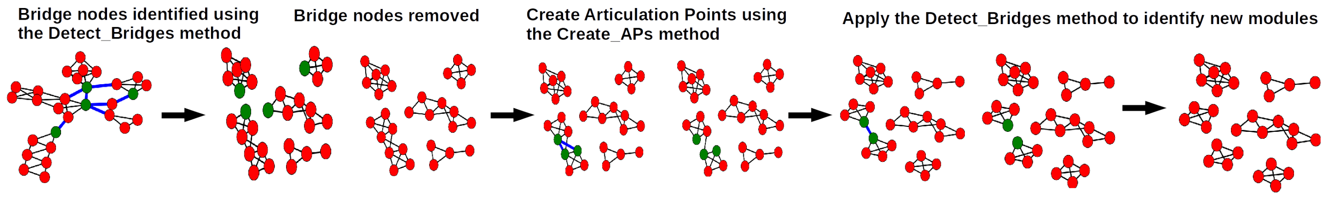

Bridge nodes identified using the Detect_Bridges method
Bridge nodes removed
Create Articulation Points using the Create_APs method
Apply the Detect_Bridges method to identify new modules

Figure 2. Running example depicting Local centrality. In the first phase, we identify weak nodes using the Detect_Bridges method. The green nodes in it represent the bridge nodes. Further, the two sub-graphs in the Bridge nodes removed phase show the results after assigning nodes to their largest component. The bridge nodes are now part of different modules. In the third phase, we apply the Create_APs method to create APs in big modules to detect new modules. The green nodes (and blue edges) represent the potential articulation points. After removing those edges, we create new APs. Finally, we again apply the Detect_Bridges method to identify new modules of the graphs.

Algorithm 1 The pseudo-code of the Merge function

```
Require: $\mathcal{L}$ =List set; $L$ = list; $\epsilon \in [0...1]$
Ensure: Set of Overlapping Lists $\mathcal{L}$
1: for $A \in \mathcal{L}$ do
2:     if $L \subseteq_\epsilon A$ then
3:         $b = L \cup A$
4:         Remove $L$ and $A$ from $\mathcal{L}$
5:         $\mathcal{L} = \mathcal{L} \cup b$
6:     end if
7: end for
8: return $\mathcal{L}$
```

$\epsilon = 1$ ensures that two lists are only merged if one of them is a proper subset of the other.

THE ALGORITHM

DCS consists of four phases: data reduction, leader selection, community expansion and merging communities.

Data Reduction

We propose a simple local search approach, local centrality, to identify the bridge nodes of the graph. The pseudo code and a running example of the local centrality is specified in Algorithm 2 and Figure 2. Initially, G is a graph that will be partitioned into modules. In Step #1 of Algorithm 2, we detect a set of articulation points (APs) whose deletion will disconnect the graph. The pseudo code of the Detect_Bridges function is specified in Algorithm 3. For each node v, we apply the $EWE(v, G)$ operation, obtaining its local graph *ego_minus_ego*. Note that we do not perform computations on $EN(v, G)$ because each node v is directly linked to all its neighbors. This leads to noise that will result in a false estimate of a node's importance. For each node v, we search for the mutual neighbors between a pair of nodes.

Algorithm 2 The pseudo-code of Local centrality

```
Require: $G$ =Graph Copy; $S$ =Max Module Size, $\epsilon \in$[0..1]
Ensure: Set of Modules $normmod$
1: $bigmod, normmod \leftarrow Detect\_Bridges(G, S)$
2: for $module \in bigmod$ do
3:     $G1 \leftarrow Create\_APs(module, \epsilon)$
4:     $bigmod, normmod \leftarrow Detect\_Bridges(G1, S)$
5: end for
6: return $normmod$
```

Algorithm 3 The pseudo-code of Detect_Bridges

```
Require: $G$ =Graph Copy; $S$ =Max Module Size
Ensure: Set of Modules ($bigmod$ and $normmod$)
1: Initialize nbr_list, $comm$, $big\_comp$={}
2: for $v \in G$ do
3:     $ego\_minus\_ego \leftarrow EWE(v, G)$
4:     for $k \in ego\_minus\_ego$ do
5:         Update nbr_list=neighbors(G, $k$)
6:         Update $comm$=Merge($comm$, nbr_list)
7:     end for
8:     Detect $big\_comp$= largest_component(comm)
9:     Use $big\_comp$ to leave node $v$ connected with its neighbors with which it shares many common neighbors and remove its edges with the loosely connected nodes.
10: end for
11: Assign modules> $S$ to $bigmod$ and else to $normmod$.
12: return $bigmod$, $normmod$
```

Even if two nodes share a single neighbor we merge them into a single group (Step #6, Algorithm 3). Once we have the list *comm* for each node v, we get its local view that explains how many distinct modules this node belongs to. In graph G, we leave node v connected to the neighbors with which it shares many mutual neighbors, i.e. where neighbors of v are also tightly connected to each other, and remove its edges with the remaining loosely connected nodes (Step #9, Algorithm 3). This process cycles over each individual node. From this step, we obtain a set of modules greater and smaller than our defined size S. The output *bigmod* contains maximal induced subgraphs that remain connected after removing any node and its adjacent edges. The output *bigmod* contains large modules, so we perform further computation on it to create articulation points (APs) that can be utilized to detect new modules. We apply Step #3 (Algorithm 2) to create articulation points in *bigmod*. There are two differences between *Create_APs* and *Detect_Bridges* methods. First, we set a small ϵ for *Create_APs*, because *bigmod* contains no articulation point. We detect nodes that might share some overlap between two modules but still are bridge nodes. A small ϵ conveys this intuition. Second, the *Create_APs* function does not result in any new module, it is mainly data reduction and only creates articulation points as shown in Figure 2. The procedure for *Create_APs* shares many similarities with Algorithm 3, we only set a small ϵ and remove Step #11 from *Detect_Bridges* function and the rest is same. This step creates a set of articulation points and returns a graph $G1$. Last, we repeat the *Detect_Bridges* procedure (Step #4, Algorithm 2) on $G1$ to identify new modules. The final output *normmod* (Step #6, Algorithm 2) contains modules less than or equal to our defined size S, while *bigmod* contains no modules and it is empty. The modules are further analyzed in the next phases to detect communities in them.

Leader Selection

Now we focus on identifying a set of leaders from each detected module. This phase is crucial, because choosing the correct leaders allows quick convergence while starting with the wrong leaders will necessitate many iterations and may get stuck in a bad local optimum. More importantly, this process should not be too expensive. We design a simple technique based on the extended degree k^+ of the nodes that identifies leaders at key positions of the module. The pseudo code of the Leader Selection function is specified in Algorithm 4. We sort the nodes in descending order using equation 2 and then, for each node v, we search for the common neighbors with the previous leaders from the leader set $\mathcal{L}$. We identify node v as a leader node if and only if at most $\epsilon\%$ of its neighbors are common with at least one of the leaders in $\mathcal{L}$. The loop cycles over each individual node to get a complete coverage of the module.

Algorithm 4 The pseudo-code of Leader Selection

```
Require: $G$ =Module Copy; $\epsilon$ =[0..1]
Ensure: Set of Leaders $\mathcal{L}$
 1: extend=Sort nodes in $G$ using equation (2)
 2: for $v \in$ extend do
 3:     Extract node_nbrs=neighbors($G$, $v$)
 4:     for lead in $\mathcal{L}$ do:
 5:         Extract lead_nbrs=neighbors(G, lead)
 6:         Compute com$\leftarrow$ common(node_nbrs, lead_nbrs)
 7:     end for
 8:     Select $v$ as a leader if where (com¡=$\epsilon$) not True
 9:     Update $\mathcal{L} \leftarrow v$
10: end for
11: return $\mathcal{L}$
```

Community Spreading Phase

Once we have a set of leading nodes, we expand the clusters around those leaders. We develop a scoring function based on conductance and internal density of the module. We consider maximum the second-order neighborhood graph of a leader. Our goal is to detect communities that possess lowest possible conductance and highest possible internal density. To find the communities around each leader L_i we use the following

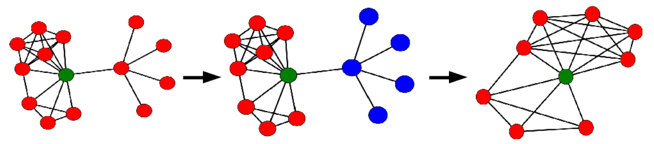


Figure 3. A simple simulation of our community spreading phase for community discovery. Step 1 shows the extracted graph around the leader (green node). Step 2 shows the nodes (blue nodes) that fail to satisfy our criteria, hence they are removed. Step 3 shows the final community structure around the leader.

equations:

$$f(G) = \frac{(c_s * w_{external})}{(2m_s * w_{internal}) + (c_s * w_{external})}, \tag{3a}$$

$$f_G^A = \underbrace{f_{G-\{A\}}}_{\text{Without A}} - \underbrace{f_{G+\{A\}}}_{\text{With A}}, \tag{3b}$$

$$D_b(A) = \begin{cases} 1, & \text{if } (D_G^{A+} - D_G^{A-}) \text{ is positive} \\ 0, & \textit{otherwise} \end{cases}, \tag{3c}$$

$$\text{Community Assignment of Node A} = f_G^A \cap D_b(A), \tag{3d}$$

where $w_{internal}$ and $w_{external}$ represent the percentage of internal and external weight (in a weighted graph) for a particular node, similarly c_s and m_s represent the number of edges joining each member of the module to the rest of the graph and number of internal edges inside the module respectively. In equation $3b$, we define our basic notion for capturing the conductance, the terms $G-\{A\}$ and $G+\{A\}$ denote the subgraphs obtained from a module without and with node A. A three step example of our approach is shown in Figure 3. The aim is to determine a subgraph around each leader L_i such that inclusion or exclusion of a node would lower the conductance score. To do so, we extract all the neighbors of L_i and then perform a loop over all the neighboring nodes to find their scoring values. Our approach is additive, where we add or subtract one node at a time. The value for f_G^A remains positive where excluding a particular node decreases the conductance, and vice versa. Note that we only consider nodes having positive f_G^A score, and if a node A turns to have negative score, it is not considered in the next iteration. One negative characteristic of conductance that can be pointed out is that it is slightly biased towards the bigger communities. Additionally, the lack of internal density information may produce weak communities, i.e., the detected communities might be huge and incoherent. To overcome this problem, we also consider the density of a community as an important measure. Basically, we generate a binary response for the graph density as defined in equation $3c$. It generates a discrete response (0 or 1). In equation $3c$, $D_b(A)$, D_G^{A+} and D_G^{A-} represent the binary density response, density of the neighborhood network with A inside and density without A respectively. The density value does not contribute to the scoring function. However, it indicates whether the inclusion of a node A inside the community is beneficial or not. Basically, if inclusion of a particular node significantly deteriorates the internal density, then it is not considered inside a community. Finally, we formalize conductance and internal density of a node A as defined in equation $3d$. Note that a node remains in the community only if conductance and density responses are positive. We summarize this intuition in Table 1. The procedure is iteratively repeated until the *second*-order neighborhood of each leader L_i is reached. However, the process stops when all the nodes in the direct neighborhood of a leader have negative values.

f_G^A	$D_b(A)$	Community Assignment
+	1	Yes
+	0	No
-	1	No
-	0	No

Table 1. Node to community assignment.

Merging Communities

The results of the previous phase are sets of communities, according to the perspective of each leader L_i. These communities are the node's social identity, they are used in the merging phase to get the global perspective of each module. Thus the next step is to merge the local communities to get a global view of each module. We use the pseudo code specified in Algorithm 1. Note that the previous phase produces large communities in big modules and small communities in small modules; setting a low ϵ for large modules produces further bigger communities and vice versa. We use different ϵ for large and small modules to handle local heterogeneity.

EXPERIMENTAL SETUP

In this section, we give an overview of the studied datasets, quality evaluation metrics and the selected CD algorithms.

Dataset Description. We test the effectiveness of our approach on eleven real-world datasets, a general overview about the features of these datasets are presented in Table 2, where: n is the number of nodes, m is the number of edges and $\bar{k}$ is the average degree of the network. We summarize the datasets as follows: **Protein Network** [1]**:** this undirected network contains protein interactions contained in yeast. A node represents a protein and an edge represents a metabolic interaction between two proteins. **Arxiv GR-QC (General Relativity and Quantum Cosmology) [17]:** it covers collaborations between authors papers submitted to General Relativity and Quantum Cosmology category. If an author i co-authored a paper with author j, the graph contains an undirected edge from i to j. **EVA**[1]**:** it contains inter-relationships networks between corporations. An edge from company X to company Y exists in the network if in company X is an owner of company Y. **US Powergrid**[1]**:** it contains information about the power grid of the Western States of the United States of America. An edge represents a power supply line. A node is either a generator, a transformer or a substation. **Dolphin social network**[1]: an undirected social network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand. **Indian Railway [6]**: nodes represent the stations, and two station s_i and s_j are joined if there exists at least one train-route such that both s_i and s_j are scheduled halts on that route. **DBLP Network[17]**: a co-authorship network where two authors are connected if they publish at least one paper together. **Amazon Network [17]**: it is mainly based on the 'Customers Who Bought This Item Also Bought' feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains an undirected edge from i to j. **Youtube Network [17]**: a friendship network of youtube users. **Road Network [17]**: a road network of California. Intersections and endpoints are represented by nodes and the roads connecting these intersections are represented by undirected edges. **Amazon Rating**[1]: it contains product ratings from the Amazon website.

[1]http://konect.uni-koblenz.de/networks/

Network	n	m	$\bar{k}$	Ground-Truth
Protein	1,458	1,993	2.73	No
Arxiv GR-QC	4,158	13,428	6.46	No
EVA	4,475	4,654	2.08	No
US Powergrid	4,941	6,594	2.67	No
Dolphin	62	159	5.13	Yes
Railway	301	1,224	8.13	Yes
DBLP	317,080	1,049,866	6.62	Yes
Amazon dataset	334,863	925,872	5.53	Yes
Youtube	1,134,890	2,987,624	5.27	Yes
Road dataset	1,965,206	2,766,607	2.91	No
Amazon rating	2,146,057	5,743,146	3.46	No

Table 2. Basic statistics of the studied Networks. We separate the datasets in two parts. We use the first four datasets to demonstrate the results of local centrality and the remaining seven for community detection algorithms.

Scoring Measures. The quality of the community is often measured by how community-like the connectivity structures of a given set of nodes are. Here we use conductance [18], cut-ratio (Cut) [10] and modularity (Mod) [19] to verify the community quality. Similarly, evaluating the quality of a partition delivered by an algorithm requires some measures to verify how similar the delivered partition is to the one required to be recovered. We use three measures, namely F1 score, Normalized Mutual Index (NMI) [16] and Adjusted Rand Index (ARI).

Selected Baselines for Comparison. We quantify the performance of our algorithm by comparing it to four state-of-the-art overlapping and non-overlapping methods, namely Demon, BigClam, NISE and Infomap.

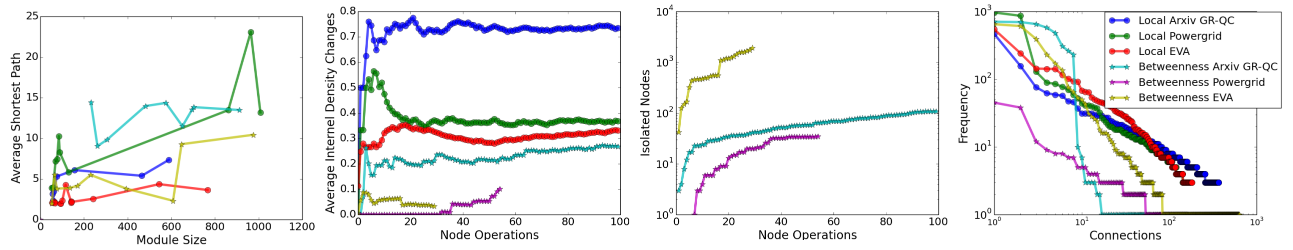
The experiments are performed on Core i7-2620M 64 bits @ 2.70GHz, equipped with 12 GB of RAM and with a kernel linux 3.2 (Ubuntu 12.04). The code for *DCS* is developed in Python.

EVALUATION RESULTS

We now present our experimental findings. We first evaluate the performance of local centrality. Further, we investigate the scalability of *DCS* on benchmark and real networks. Next, we evaluate the quality of the delivered communities using their conductance, cut-ratio and modularity scores. We also perform the quantitative analysis of the delivered results by comparing it against the ground truth data. Last, we present our results for automatic algorithm selection.

Evaluating Modular Output

The main objective of this section is to demonstrate the potential of local centrality to identify nodes that are positioned at critical positions between different modules in a graph. We evaluate the modular output from four networks, namely Protein, Arxiv GR-QC, EVA and US Powergrid networks. For



(a) APLs (b) ID Changes (c) Singleton Changes (d) Distribution of Module Size

Figure 4. Analysis on the Arxiv GR-QC, US Powergrid and Eva networks for local centrality and betweenness centrality (see legend in 4(d)). (a) the average path lengths (APLs) for the modules, (b and c) the internal density (ID) changes and the average isolated nodes for each node operation, and (d) the module size distribution.

proper assessment of local centrality, we select dense and sparse networks regarding their internal densities and number of edges inside a graph. In order to investigate the topological locality of the nodes selected by local centrality of the graph, several network properties, e.g., average shortest paths inside the modules, average internal density changes, average isolated nodes and distribution of module size are analyzed and compared. We choose betweenness centrality as its competitor, because it is one of the most accurate measures to detect bridging nodes. One shall note that we are primarily interested in showing that the results from local centrality are nearly comparable to (and sometimes better than) betweenness centrality.

Figure 4 shows the results for the selected datasets.For betweenness centrality, we sequentially remove the nodes after each recalculation to observe the changes in the network properties. As a first evidence of local centrality's strong performance, we observe the topological properties of modules detected by local centrality from an average path length (APL) point of view. Figure 4(a) shows the APL inside each module of the graph. We compute the average shortest path for modules with 100 or more nodes. The APL distinguishes an easily negotiable network from one that is complicated and inefficient, with a shorter APL being more desirable. Further, a small module with higher APL is a bad and inefficient separation of the graph. Overall, we obtain strong results from local centrality that are comparable and in some cases even superior to betweenness centrality regarding the shortest paths. For the EVA dataset, the modules identified by local centrality have lower APLs as compared to betweenness centrality even for bigger modules. We notice only one module from the US Powergrid network where a module identified by local centrality has a high APL. The results for betweenness centrality on the Powergrid network are not shown in 4(a), because it fails to detect any modules and produces a huge network of 3,695 nodes. Further, the APLs for betweenness centrality are very high on the Arxiv GR-QC dataset, on the contrary local centrality identifies a set of modules with lower APLs for the same dataset. Figure 4(b) and 4(c) compare the trends for internal density (ID) changes of the graph and the isolated nodes as a consequence of performing sequential node operations. The combination of the two explains some interesting trends. We notice that local centrality initially identifies small modules that increase the average ID; however, after some iterations it starts producing big modules. Further, the average ID changes for local centrality show a very consistent behavior at each iteration, this shows that the big modules identified by local centrality have high internal densities. In other words, local centrality enhances the modularity of the network, because it identifies nodes that are located between modules and not at the center of modules. We get no isolated nodes from local centrality; however, we obtain many isolated nodes from betweenness centrality. Local centrality selects the nodes that are sparsely connected in the graph; however, betweenness centrality normally identifies the high degree (and high clustering) nodes that have many peripheries connected to them, that results in smaller modules, many isolated nodes and low internal density. For local centrality, we only observe a drop in the internal density for the Powergrid dataset. Figure 4(d) shows the module size distribution on three datasets. For local centrality, the tail of distribution appears power law like. There are very few small modules. The largest deviation from the power law is in the tail, because we do not have singletons or very small modules in the data. For betweenness centrality, we observe some big modules followed by a chain of very small modules. For betweenness centrality, we notice a monotonously decreasing distribution for the Arxiv GR-QC network. Finally, we report more results in Table 3. Local centrality runs at least 1,000 times faster than betweenness centrality. For the Powergrid dataset, betweenness centrality returns a huge network of 3,596 nodes that results in a slightly bigger average size.

	Local Centrality			Betweenness Centrality		
	$\lvert M_s \rvert$	Size	Time	$\lvert M_s \rvert$	Size	Time
Protein	82	17.78	**1s**	239	5.95	3m:10s
Arxiv GR-QC	369	11.27	**5s**	202	20.89	3h:20m
EVA	183	24.46	**3s**	632	7.05	46m
US Powergrid	129	38.30	**3s**	93	**53.11**	3h:15m

Table 3. Comparative analysis between local and betweenness centrality, where $|M_s|$ and Size represent the number of identified modules and average module size.

Scalability of DCS

We compare the scalability of DCS with other baselines by measuring the runtime on networks of increasing sizes. We generate the LFR benchmark networks [15] varying the number of nodes from 1,000 to 400,000 with the average degree fixed. In Demon, we set ϵ=0.4, which provides best results [8]. In BigClam, we use the default settings provided in the code. For DCS, we have two main parameters, namely the module size $|S|$ and overlapping factor ϵ to merge local com-

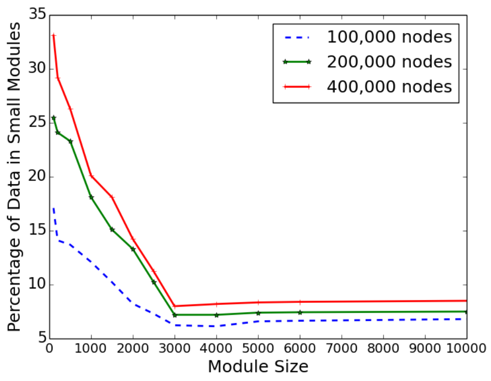


Figure 5. Analysis on a sample of three benchmark networks for local centrality. We obtain optimum results for —S— =3,000, because we have the lowest percentage of data confined in small modules.

munities. We vary the two parameters and see what happens to the number of modules, number of communities and quality of the results. We summarize our observations for module size $|S|$ in Figure 5. We notice that a low $|S|$ produces many small modules (of size ≤3 over at least 18% of the network) that disturb the network topology. We find strong results for $|S|$=3,000. For $|S| > 3000$, we can see that for all three networks the disturbance percentage remains almost constant (very slow slight upward trend). We observed similar trend for all the tested networks. Once we detect the proper modules, the ϵ was not difficult to find. We find optimum results for ϵ_{large}=0.75 and ϵ_{small}=0.30. Figure 6 shows the runtime results for each algorithm. Demon scales to networks of around 100,000 nodes and then its runtime becomes prohibitively large. Similarly, BigClam's runtime also escalates with increasing network size. In contrast to that DCS can process networks with hundreds and thousands of nodes within few minutes (three minutes for a network with 100,000 nodes). We also test the scalability of DCS on real world networks as reported in Table 4. DCS is remarkably faster than all other methods, especially compared to other overlapping methods. Note that we are not able to provide the results for the Road network and Amazon rating datasets on Demon: for that it failed to scale up on these networks.

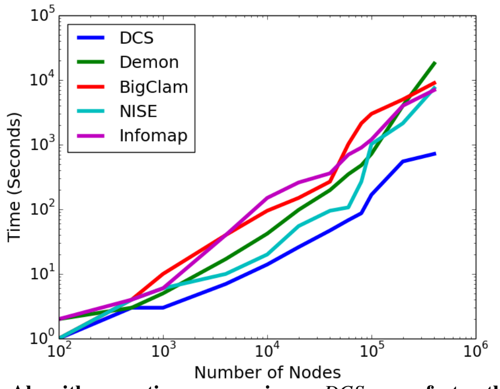


Figure 6. Algorithm runtime comparison. DCS **runs faster than all competing baselines.**

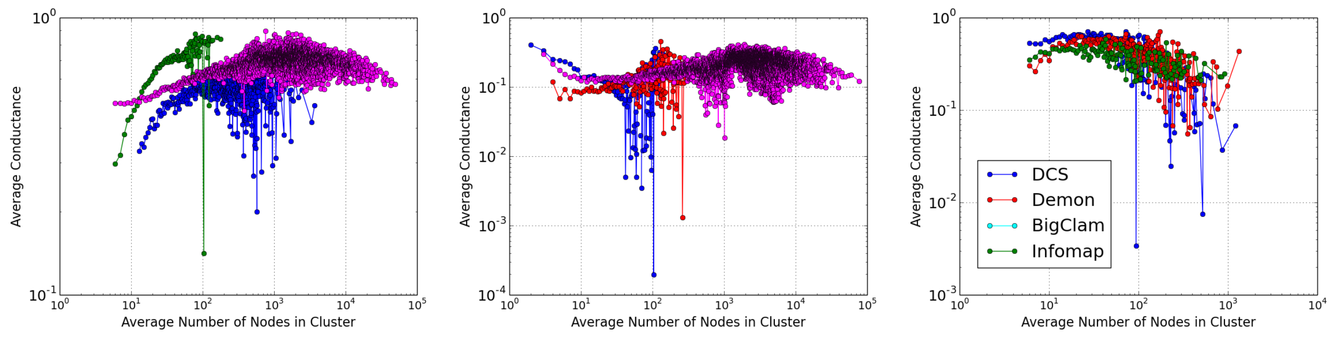
	DCS	Demon	BigClam	NISE	Infomap
DBLP	**8min**	360min	150min	65min	90min
Amazon	**12min**	840min	110min	125min	120min
CRN	**29min**	-	280min	260min	300min
Rating	**75min**	-	340min	365min	390min

Table 4. Approximate running time for different methods on three datasets. DCS **is clearly the fastest, even on large datasets.**

Reasons for DCS Scalability. It is important to have an understanding of what makes DCS more scalable as compared to other existing methods, especially $NISE$ and Demon. $NISE$ applies biconnected operation to extract the core of the graph. The biconnected operation only detaches whiskers and delivers the largest connected component that contains more than 80% of the nodes from the graph. For DCS, the modular approach speeds up the algorithm and is thus essential for making it scalable to large networks. The output from data reduction phase contains not a single module, but a list of modules (≤ our defined size). For $NISE$, we notice that the community expansion phase takes a long time, which for DCS is very fast. On Amazon dataset, we notice that the community expansion takes five minutes and two hours for DCS and $NISE$ respectively. Further, DCS and Demon both use the same merge function, but there is an important difference between how they use it. Demon repeats the merging procedure for every node of the graph, which is the biggest hurdle for its scalability. In contrast to that, DCS only merges communities expanded around the selected leaders (and only inside each module of size S). These factors make DCS more scalable as compared to these methods.

Community Quality via Scoring Functions

We evaluate the quality of communities by computing the average conductance for each community size. A quality algorithm should return medium size communities that cover a large portion of the graph with minimum conductance. Figure 7 shows the conductance-vs-size for the CD algorithms on three datasets. To clearly demonstrate the significant results, we only select the three best algorithms for each dataset For each method, we sort the clusters based on the size, and then take the average conductance for each size. Note that lower conductance indicates better quality of communities. Basically, we are interested in evaluating the quality of the delivered communities and how community size effects the overall conductance. We plot on log-log scale. The x-axis and y-axis represent the size of communities and average conductance for that community size. Note that we are not able to provide the results for the road dataset on Demon, because it failed to scale up for it. As seen in the plots, DCS clearly performs better than the other methods on all three datasets. Further, Leskovec et al. [18] have found that the communities between 100 and 1,000 nodes have the lowest conductance. For DCS, we notice best possible communities with approximately 100-200 nodes as shown in Figure 7, which shows that DCS delivers communities that have low conductance and are more community-like. Demon and BigClam produce communities with higher conductance for the DBLP and Amazon datasets respectively, we omit the results as they were not competitive. $NISE$ produces huge size communities with higher conductance on all three dataset as shown in the figures. Similarly, BigClam produces communities with



(a) DBLP (b) Amazon (c) Road

Figure 7. Analysis on the DBLP, Amazon and road networks for CD algorithms. Each color represents a CD algorithm (see legend in 7(c)). The plots show community size vs. average conductance with a lower curve indicating better performance. Overall, DCS performs better than all other methods for all three datasets. Further, we notice a perfect dip (low conductance communities) for DCS for communities with around 80-200 nodes.

	DCS				Demon				BigClam				NISE				Infomap			
	$\lvert C\rvert$	$\lvert\bar{c}\rvert$	Mod	Cut	$\lvert C\rvert$	$\lvert\bar{c}\rvert$	Mod	Cut	$\lvert C\rvert$	$\lvert\bar{c}\rvert$	Mod	Cut	$\lvert C\rvert$	$\lvert\bar{c}\rvert$	Mod	Cut	$\lvert C\rvert$	$\lvert\bar{c}\rvert$	Mod	Cut
DBLP	63,289	50	0.34	0.18	8,954	10	0.24	0.20	25,000	22	0.27	0.34	26,503	463	0.29	0.19	14,702	22	0.21	0.28
Amazon	38,501	12	0.23	0.10	25,517	19	0.10	0.14	25,000	24	0.16	0.27	27,763	912	0.22	0.11	15,747	22	0.18	0.17
Road	342,581	14	0.15	0.23	-	-	-	-	20,000	53	0.09	0.28	14,000	747	0.03	0.19	10,142	194	0.10	0.25

Table 5. Comparative analysis of DCS with other community detection algorithms, where $|C|$, $|\bar{c}|$, Mod and Cut represent the number of communities, average community size, modularity and cut-ratio for each method. For Modularity a high score and for cut-ratio a low score represent a better result.

higher conductance for the Amazon dataset. Further, we report the number of communities, average community size, modularity and cut-ratio scores in Table 5. To calculate the modularity for each community of the network is prohibitive. For time and memory constraints, we chose 400 randomly selected communities and calculate their modularity. Also in this case, DCS was able to easily outperform all the other algorithms. We can conclude that DCS with a manageable number of medium-sized communities is able to outperform more complex methods by producing communities of better quality.

Evaluation using Ground Truth

We have the ground truth data for the Dolphin, railway, DBLP, Amazon and Youtube networks, thus for these networks, we compare the delivered communities against the ground truth data. In the Dolphin dataset, each community represents the gender of the dolphins. In the railway dataset, each community represents the number of trains within each state. In the DBLP dataset, each publication venue corresponds to a ground-truth community. In the Amazon dataset, each ground truth community corresponds to a product category that Amazon provides. In the Youtube dataset, each ground-truth community corresponds to a user-defined group.

Table 6 reports the $F1$, NMI and ARI scores on five datasets. Overall, DCS gives superior overall performance. This further means that, while DCS is more scalable and delivers lower conductance communities than other competing baselines, it also achieves superior results regarding the quality of detected communities. On all five networks, DCS achieves the best results. The only exception is the F1-score for the Amazon dataset. For the Amazon dataset, $NISE$ was able to outscore DCS in $F1$ score; however, DCS has a higher NMI and ARI scores for the same dataset. There is an important distinction to be made about these results: For the $NISE$ algorithm, the number of communities k needs to be manually defined. We select the number of clusters defined by the authors. However, we vary the number of communities k and observe what happens to the overall results. We notice that for higher k the running time increases and the $F1$ score also starts to drop. DCS requires no such parameters, it determines the number of communities automatically. Further, for the DBLP and Youtube datasets, DCS outperforms all other methods with 0.23 and 0.13 for $F1$, which is at least 22% higher than any other method. This shows that DCS is able to perform equally well on large graphs.

Algorithm Selection

Generally, it is neither clear how to choose among existing approaches nor is it clear when a particular algorithm performs better than the others. Moreover, it is widely accepted for any machine learning solution that there is no 'one-size-fits-all' and each solution excels in a specific part of the problem space. We propose a simple model for the algorithm selection problem as illustrated in Figure 8. We train a supervised classifier on a set of CD algorithms using the LFR benchmark. There are not many real world networks available with explicit ground truth, so we rely on the LFR benchmark. The usage of the benchmark provides a facilitation as we can define a known community structure and evaluate how well the algorithm is able to detect it by comparing it against the ground truth. Of course there are some limitations: there are many different community definitions and benchmark networks can only capture a few. However, the LFR benchmark is known to generate networks that possess strong properties of real complex networks. The parameter choices for the benchmark are the following: number of nodes (n), average and maximum degree ($\bar{k}$ and k_{max}), minimum and maximum community size (Min_c and Max_c), number of overlapping nodes (on) and the mixing parameters (μ) for the topology and weight. The mixing parameters represent the ratio between the external con-

	DCS			Demon			BigClam			NISE			Infomap		
	F1	NMI	ARI	F1	NMI	ARI	F1	NMI	ARI	F1	NMI	ARI	F1	NMI	ARI
Dolphin	**0.90**	**0.56**	**0.85**	0.62	0.40	0.41	0.001	0.20	0.42	0.29	0.52	0.48	0.56	0.40	0.63
Railway	**0.35**	**0.21**	**0.89**	0.20	0.15	0.56	0.31	0.16	0.80	0.17	0.16	0.32	0.33	0.11	0.82
DBLP	**0.23**	**0.19**	**0.34**	0.14	0.07	0.12	0.16	0.13	0.23	0.18	0.12	0.26	0.11	0.10	0.17
Amazon	0.30	**0.20**	**0.39**	0.23	0.18	0.29	0.26	0.12	0.31	**0.48**	0.11	0.35	0.20	0.14	0.26
Youtube	**0.13**	**0.11**	**0.18**	0.045	0.008	0.009	0.08	0.05	0.11	0.10	0.08	0.14	0.002	0.001	0.005

Table 6. Scores of the validation metrics with respect to the ground truth communities. A high score indicates better results. Overall, *DCS* performs better than other baselines.

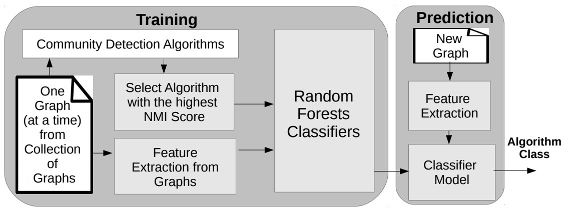

Training
Community Detection Algorithms
One Graph (at a time) from Collection of Graphs
Select Algorithm with the highest NMI Score
Feature Extraction from Graphs
Random Forests Classifiers
Prediction
New Graph
Feature Extraction
Classifier Model
Algorithm Class

Figure 8. Model for automatic algorithm selection.

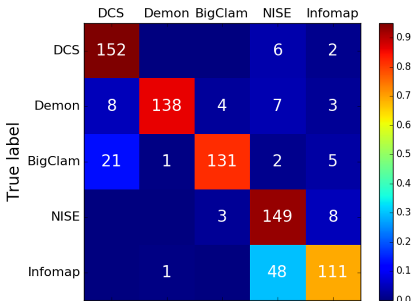


Figure 9. Confusion matrix for 10% test networks. We achieve accuracy of 0.85.

nections of a node to its degree. For experiments, we generate 8,000 networks with the following (in these ranges) configurations: network size (n={40, 3000}), Min_c and Max_c (c={10, 3000}), average degree ($\bar{k}$={8, 35}) and mixing parameters (μ_t={0.1,1}, μ_w={0.1,1}). The maximum network size and community size correspond to the maximum module size delivered by local centrality. We already discussed that the local centrality delivers best results when the maximum module size is below 3000. Note that we only generate unweighted and undirected networks that are either non-overlapping or overlapping. Next, we studied various micro and macro properties that are used to describe network topology: transitivity, degree distribution, eccentricity, diameter, centrality, graph density and average clustering coefficient (CC) are a few of them. The challenge is to select features that describe the way the community structures interact without explicitly knowing the community structures. We extract two features that describe the network topology and the community structure in the best possible way, namely graph density and average CC. We discussed the motivation behind the graph density in Background and Community Spreading sections. The CC shows the extent to which nodes in a graph tend to cluster together, it describes how complete the neighborhood of a node is. Further, [20, 15] have discussed that the average CC, taken over all the nodes, has a strong correlation with the topological mixing parameter μ_t. Similarly, the weighted version of the CC has a strong correlation with the weighted mixing parameter μ_w. We calculate CC using the following equation:

$$c^{(v)} = \frac{2T(v)}{deg(v)(deg(v-1))} \sum_{vu} (\grave{w}_{vu}\grave{w}_{vw}\grave{w}_{uw}), \qquad (4)$$

where $T(v)$ is the number of triangles through node v and $\grave{w}$ represents the normalized edge weights. The second part of equation 4 is only for weighted graphs. We take the average CC over all the nodes in G. For classification purposes, we choose Random Forests Classifiers [5] because of two important reasons. First, they do not require parameter tuning, second, they do not require the specification of a feature space, as Support Vector Machines (SVMs) require the kernel settings. In the training phase, we extract the selected measures from each graph, and apply CD algorithms (DCS, Demon, BigClam, $NISE$ and Infomap) on it. We evaluate the quality of the communities using the NMI scores. The algorithm with the highest score, i.e. the algorithm that performs best on that particular graph, is assigned to a class. We split the data into random training and testing subsets with 90% for training and 10% for testing. We use 10 fold cross validation to train the classifier. Figure 9 shows the confusion matrix for the testing phase. We obtain a score of 0.85 for accuracy rate and 0.15 for error rate. From these results, we can see that even with simple network features it is possible to obtain an accurate prediction of the algorithm class. Last, after training a classifier, we again apply the networks studied in the previous section to predict their algorithm class. We notice that for two networks (Dolphin and railway), the classifier selects DCS because it produces best results for these datasets. For the remaining large networks (DBLP, Amazon and Youtube datasets), we initially identify modules using local centrality and then we extract a set of features from each module and allow the trained classifier to predict the algorithm class for each module. Table 7 shows the new results for three datasets after applying our algorithm selection model. The output shows at least 20% improvement in overall results as compared to the results each algorithm separately delivers. These results further mean that by training a classifier on a large collection of graphs (and CD algorithms), we can model it to predict an appropriate CD algorithm based on a specific graph structure that can improve the overall results.

	F1	NMI	ARI
DBLP	0.33	0.29	0.44
Amazon	0.45	0.37	0.52
Youtube	0.19	0.15	0.25

Table 7. Scores of the validation metrics after applying our algorithm selection model

CONCLUSIONS AND FUTURE WORK

In this work, we introduced a bridge based community detection method (*DCS*) that is scalable and accurately discovers the overlapping communities from real world networks. We proposed a modular approach, where we first detect large modules of the graph and then analyze each module independently to discover community structures in it. From our experiments we observed that local centrality, our approach to identify modules of the graph, produces comparable and in some cases even better results than betweenness centrality. We have shown that *DCS* is scalable to large networks and requires less than three minutes for a graph with 100,000 nodes. Experiments have shown that *DCS* delivers better community structures w.r.t state-of-the-art algorithms, evaluated using their conductance, cut-ratio and modularity scores. We also evaluated the quantitative performance of *DCS* on real world networks with their actual community structures. Our results have shown that *DCS* uncovers the community structures better than other methods both on small and large networks. Finally, we presented our solution for algorithm selection of the graph. Our results have shown that it is possible to select the best algorithm based on the properties of the network alone. Moreover, we have shown that our model has improved the overall results as compared to the results each algorithm separately delivers.

As an immediate future work, we plan to study the asymptotic properties of the local centrality, improving local centrality to avoid detection of small modules in the initial phases and the complexity analysis of *DCS*. We also plan to further study the algorithm selection procedure by considering a more comprehensive set of algorithms.

REFERENCES

1. Charu C. Aggarwal, Yan Xie, and Philip S. Yu. Towards community detection in locally heterogeneous networks. In *SDM*, pages 391–402, 2011.
2. Reid Andersen and Kevin J. Lang. Communities from seed sets. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 223–232, New York, NY, USA, 2006. ACM.
3. Maria Balcan and Yingyu Liang. Modeling and detecting community hierarchies. In *SIMBAD*, 2013.
4. Vincent D Blondel, Jean loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks, 2008.
5. Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.
6. Tanmoy Chakraborty, Sriram Srinivasan, Niloy Ganguly, Animesh Mukherjee, and Sanjukta Bhowmick. On the permanence of vertices in network communities. In *KDD*, 2014.
7. Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 2004.
8. Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Demon: A local-first discovery method for overlapping communities. In *KDD*, 2012.
9. Easley David and Kleinberg Jon. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, New York, NY, USA, 2010.
10. Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.
11. Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1978.
12. M. Girvan and Mark E. J. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, June 2002.
13. Steve Gregory. A fast algorithm to find overlapping communities in networks. In *ECML PKDD*. 2008.
14. Woochang Hwang, Taehyong Kim, Murali Ramanathan, and Aidong Zhang. Bridging centrality: Graph mining from element level to group level. In *KDD*, 2008.
15. Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 2009.
16. Andrea Lancichinetti, Santo Fortunato, and Janos Kertesz. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.
17. Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
18. Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. In *WWW*, 2008.
19. M E Newman. Modularity and community structure in networks. *Proc Natl Acad Sci U S A*, 103(23):8577–8582, June 2006.
20. Leto Peel. Estimating network parameters for selecting community detection algorithms. In *Fusion*, 2011.
21. Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.
22. Bradley S. Rees and Keith B. Gallagher. Overlapping community detection by collective friendship group inference. In *ASONAM*, 2010.

23. Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, January 2008.

24. Joyce Jiyoung Whang, David F. Gleich, and Inderjit S. Dhillon. Overlapping community detection using seed set expansion. In *CIKM*, New York, NY, USA, 2013.

25. Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 2013.

26. Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *WSDM*, 2013.