Overlapping Community Detection Using Seed Set Expansion

Joyce Jiyoung Whang
Dept. of Computer Science
University of Texas at Austin
joyce@cs.utexas.edu

David F. Gleich
Dept. of Computer Science
Purdue University
dgleich@purdue.edu

Inderjit S. Dhillon
Dept. of Computer Science
University of Texas at Austin
inderjit@cs.utexas.edu

ABSTRACT

Community detection is an important task in network analysis. A community (also referred to as a cluster) is a set of cohesive vertices that have more connections inside the set than outside. In many social and information networks, these communities naturally overlap. For instance, in a social network, each vertex in a graph corresponds to an individual who usually participates in multiple communities. One of the most successful techniques for finding overlapping communities is based on local optimization and expansion of a community metric around a seed set of vertices. In this paper, we propose an efficient overlapping community detection algorithm using a seed set expansion approach. In particular, we develop new seeding strategies for a personalized PageRank scheme that optimizes the conductance community score. The key idea of our algorithm is to find good seeds, and then expand these seed sets using the personalized PageRank clustering procedure. Experimental results show that this seed set expansion approach outperforms other state-of-the-art overlapping community detection methods. We also show that our new seeding strategies are better than previous strategies, and are thus effective in finding good overlapping clusters in a graph.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*

General Terms

Algorithms, Experimentation

Keywords

Clustering, Community Detection, Overlapping Clusters, Seeds, Seed Set Expansion

1. INTRODUCTION

In many social and information networks, nodes participate in multiple communities. For instance, in a social network, a node's communities correspond to its social circles [18]. We study the problem of overlapping community detection to find these groups. More specifically, we investigate how to select the seed sets in a method for overlapping community detection that grows communities around seeds. These local expansion methods are among the most successful strategies for overlapping community detection [26]. However, principled methods to choose the seeds are few and far between. When they exist, they are usually computationally expensive, for instance, using maximal cliques as seeds [23]. Empirically successful strategies include exhaustively exploring all individual seeds and greedy methods that randomly pick a vertex, grow a cluster, and continue with any unassigned vertex. The goal of traditional, exhaustive clustering is to determine a cluster for each and every data point or node. In the community detection problem, instead, we wish to relax this goal and allow incomplete coverage of the network. Put another way, the data may not support assigning a node to a community and we want to respect that feature in our output.

The seeding strategies we consider are based on the same distance kernel that underlies the equivalence between kernel k-means and spectral clustering [9]. Using this distance function, we can efficiently locate a good seed *within* an existing set of vertices of the graph. In particular, a strategy we propose involves computing many clusters using a multilevel weighted kernel k-means algorithm on the graph (the Graclus algorithm) [9]. We use the corresponding distance function to compute the "centroid vertex" of each cluster and then use the neighborhood set of that centroid vertex as the seed region for community detection. This strategy is inspired by recent work on finding the best communities in a network using a carefully selected set of vertex neighborhoods as seeds [11]. These seeds were centers within their respective vertex neighborhoods. In this paper, we take this idea further and consider using the centers of larger regions.

The algorithm we use to grow a seed is based on personalized PageRank random walks, which we explain further in Section 4.3. The full algorithm to compute overlapping clusters from the seeds is discussed in Section 4. The algorithm begins by filtering out regions of the graph that will not participate in an overlapping clustering. We run the seed finding algorithm and the seed expansion method on the filtered graph. We then post-process this output to assign communities for the vertices removed by filtering. We show that a simple propagation of our communities to the removed regions does not increase the underlying objective function. The main contributions of this paper are:

- we propose using a kernelized distance function to determine seeds for an overlapping community detection strategy based on a high quality partitioning;

CIKM '13, October 27 - November 1, 2013, San Francisco, USA.
ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505535.

- we find that using a seeding strategy based on the centroids of communities returned from Graclus, a high quality graph partitioning scheme [9], outperforms other strategies for choosing the seeds;
- we find that an independent set of high-degree vertices, a strategy we call "spread hubs", performs almost as well as the Graclus clusters in terms of accuracy;
- we also find that all of the seed set expansion strategies significantly outperform existing methods in identifying a set of known communities.

Our method scales to problems with over 45 million edges, whereas other state of the art methods for overlapping clustering were unable to complete on these large datasets.

2. RELATED WORK

2.1 Overlapping community detection

We summarize a few closely related methods from a recent survey [26]. The method we employ is called local optimization and expansion. Starting from a seed, such a method greedily expands a community around that seed until it reaches a local optima of the community detection objective. In our case, we use a personalized PageRank based cut finder for the local expansion method (see Section 4.3). Other overlapping community detection methods include line graph partitioning, clique percolation, eigenvector methods, egonet analysis, and low-rank models. Given a graph $G = (\mathcal{V}, \mathcal{E})$, the line graph of $L(G)$ – also known as the dual graph – has a vertex for each edge in G and an edge whenever two edges (in G) share a vertex. For instance, the line graph of a star is a clique. A partitioning of the line graph induces an overlapping clustering in the original graph [3]. Clique percolation methods look for overlap between fixed size cliques in the graph [20]. Clique based techniques often fail to scale to large networks. Eigenvector methods generalize spectral methods and use a soft clustering scheme applied to eigenvectors of the normalized Laplacian or modularity matrix in order to estimate communities [28]. Egonet analysis methods use the theory of structural holes [7], and compute and combine many communities through manipulating egonets [22, 8]. We compare against the Demon method [8] that uses this strategy. Finally, we note that other low-rank methods such as non-negative matrix factorizations [15, 27] identify overlapping communities as well. We compare against the Bigclam method [27] that uses this approach.

2.2 Seeding strategies

Determining how to seed a local expansion method is, arguably, the critical problem within these methods. Strategies to do so include using maximal cliques [26], prior information [10], or locally minimal neighborhoods [11]. The latter method was shown to identify the vast majority of *good* conductance sets in a graph; however, there was no provision made for total coverage of all vertices.

2.3 Graph kernels and distances

The foundation of our new seeding strategy is a distance kernel between vertices. Our choice of kernel underlies the relationship between kernel k-means and spectral clustering [9]. Many other graph kernels involve the exponential or inverse of a matrix [14]. Any such kernel with an efficient means of computing distances would suffice.

3. PRELIMINARIES

In this section, we discuss the overlapping community detection problem, and review some traditional metrics for graph clustering.

3.1 Problem statement

Given a graph $G = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, the graph clustering problem is to partition the graph into k disjoint clusters $\mathcal{C}_1, \cdots, \mathcal{C}_k$ such that $\mathcal{V} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k$. While graph clustering traditionally finds exhaustive and disjoint clusters, the overlapping community detection problem is to find overlapping clusters that are not necessarily exhaustive. Formally, we seek k overlapping clusters such that $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k \subseteq \mathcal{V}$. Throughout the paper, the terms *set*, *cluster*, and *community* are used interchangeably.

3.2 Measures of cluster quality

We can represent a graph with n vertices as an $n \times n$ adjacency matrix A such that $A_{ij} = e_{ij}$ where e_{ij} is the weight of an edge between vertices i and j, or $A_{ij} = 0$ if there is no edge. We assume that all the graphs are undirected graphs, i.e., A is symmetric. Let us define $\texttt{links}(\mathcal{C}_p, \mathcal{C}_q)$ to be the sum of edge weights between vertex sets $\mathcal{C}_p$ and $\mathcal{C}_q$.

Now, we review some popular measures for gauging the quality of clusters: cut, conductance, and normalized cut.

Cut. The cut of cluster $\mathcal{C}_i$ is defined as the sum of edge weights between $\mathcal{C}_i$ and its complement, $\mathcal{V} \backslash \mathcal{C}_i$. That is,

$$\texttt{cut}(\mathcal{C}_i) = \texttt{links}(\mathcal{C}_i, \mathcal{V} \backslash \mathcal{C}_i). \tag{1}$$

Conductance. The conductance of a cluster is defined to be the cut divided by the least number of edges incident on either set $\mathcal{C}_i$ or $\mathcal{V} \backslash \mathcal{C}_i$:

$$\texttt{cond}(\mathcal{C}_i) = \frac{\texttt{links}(\mathcal{C}_i, \mathcal{V} \backslash \mathcal{C}_i)}{\min\left(\texttt{links}(\mathcal{C}_i, \mathcal{V}), \texttt{links}(\mathcal{V} \backslash \mathcal{C}_i, \mathcal{V})\right)}.$$

By definition, $\texttt{cond}(\mathcal{C}_i) = \texttt{cond}(\mathcal{V} \backslash \mathcal{C}_i)$. The conductance of a cluster is the probability of leaving that cluster by a one-hop walk starting from the smaller set between $\mathcal{C}_i$ and $\mathcal{V} \backslash \mathcal{C}_i$.

Normalized Cut. The normalized cut of a cluster is defined by the cut with volume normalization as follows:

$$\texttt{ncut}(\mathcal{C}_i) = \frac{\texttt{links}(\mathcal{C}_i, \mathcal{V} \backslash \mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}. \tag{2}$$

Notice that $\texttt{ncut}(\mathcal{C}_i)$ is always lesser than or equal to $\texttt{cond}(\mathcal{C}_i)$.

3.3 Graph clustering and weighted kernel k-means

It has been shown that a weighted graph clustering objective is equivalent to a weighted kernel k-means objective [9]. For example, for the exhaustive graph clustering problem, the normalized cut objective of a graph G is

$$\texttt{ncut}(G) = \min_{\mathcal{C}_1, \ldots, \mathcal{C}_k} \sum_{i=1}^{k} \frac{\texttt{links}(\mathcal{C}_i, \mathcal{V} \backslash \mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}. \tag{3}$$

This objective can be shown to be equivalent to a weighted kernel k-means objective by defining a weight for each data point (vertex) to be the degree of the vertex, and the kernel matrix to be $K = \sigma D^{-1} + D^{-1} A D^{-1}$, where D is the diagonal matrix of degrees (i.e., $D_{ii} = \sum_{j=1}^{n} A_{ij}$), and σ is a scalar typically chosen to make K positive-definite. Then,

we can quantify the distance between a vertex $v \in \mathcal{C}_i$ and cluster $\mathcal{C}_i$, denoted $\texttt{dist}(v, \mathcal{C}_i)$, as follows:

$$\texttt{dist}(v, \mathcal{C}_i) = \quad (4)$$
$$-\frac{2\,\texttt{links}(v, \mathcal{C}_i)}{\texttt{deg}(v)\,\texttt{deg}(\mathcal{C}_i)} + \frac{\texttt{links}(\mathcal{C}_i, \mathcal{C}_i)}{\texttt{deg}(\mathcal{C}_i)^2} + \frac{\sigma}{\texttt{deg}(v)} - \frac{\sigma}{\texttt{deg}(\mathcal{C}_i)}$$

where $\texttt{deg}(v) = \texttt{links}(v, \mathcal{V})$, and $\texttt{deg}(\mathcal{C}_i) = \texttt{links}(\mathcal{C}_i, \mathcal{V})$.

3.4 Datasets

We use eight different real-world networks from [1], [19], [24]. The networks are presented in Table 1. All the networks are connected, undirected graphs.

Collaboration networks. In a collaboration network, vertices indicate authors, and edges indicate co-authorship. If authors u and v wrote a paper together, there exists an edge between them. So, if a paper is written by k authors, this is represented by a k-clique in the network. HepPh, AstroPh, and CondMat networks are constructed based on the papers submitted to High Energy Physics (Phenomenology) category, Astrophysics category, and Condensed Matter Physics category under the arXiv e-print service, respectively. The DBLP network is constructed based on the DBLP computer science bibliography website.

Social networks. In a social network, vertices represent individuals and edges represent social interactions between them. Flickr is an online photo sharing application, Myspace is a social entertainment networking service, and LiveJournal is a blogging application where users can publish their own journals. Users can make a friendship relationship with each other in each of these websites.

Product network. In the Amazon product network, vertices represent products and edges represent co-purchasing information. If products u and v are frequently co-purchased, there exists an undirected edge between them.

Table 1: Summary of networks

Graph	No. of vertices	No. of edges
Collaboration networks		
HepPh	11,204	117,619
AstroPh	17,903	196,972
CondMat	21,363	91,286
DBLP	317,080	1,049,866
Social networks		
Flickr	1,994,422	21,445,057
Myspace	2,086,141	45,459,079
LiveJournal	1,757,326	42,183,338
Product network		
Amazon	334,863	925,872

4. OVERALL ALGORITHM

Now, we explain the overall algorithm which consists of four phases: filtering, seeding, seed set expansion, and propagation. In the filtering phase, we remove regions of the graph that are trivially separable from the rest of the graph, so will not participate in overlapping clustering. In the seeding phase, we find seeds in the filtered graph, and in seed set expansion phase, we expand the seed sets using a personalized PageRank clustering scheme. Finally, in the propagation phase, we further expand the communities to the regions that were removed in the filtering phase.

4.1 Filtering Phase

The goal of the filtering phase is to identify regions of the graph where an algorithmic solution is required to identify the overlapping clusters. To explain our filtering step, recall that almost all graph partitioning methods begin by assigning each connected component to a separate partition. Any other choice of partitioning for disconnected components is entirely arbitrary. The Metis procedure [12], for instance, may combine two disconnected components into a single partition in order to satisfy a balance constraint on the partitioning. For the problem of overlapping clustering, an analogous concept can be derived from biconnected components. Formally, a biconnected component is defined as follows:

DEFINITION 1. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, a biconnected component is a maximal induced subgraph $G' = (\mathcal{V}', \mathcal{E}')$ that remains connected after removing any vertex and its adjacent edges in G'.*

Let us define the size of a biconnected component to be the number of edges in G'. Now, consider all the biconnected components of size one. Notice that there should be no overlapping partitions that use these edges because they bridge disjoint communities. Consequently, our filtering procedure is to find the largest connected component of the graph after we remove all single-edge biconnected components. We call this the "biconnected core" of the graph even though it may not be biconnected. Let $\mathcal{E}_S$ denote all the single-edge biconnected components. Then, the biconnected core graph is defined as follows:

DEFINITION 2. *The biconnected core $G_C = (\mathcal{V}_C, \mathcal{E}_C)$ is the maximum size connected subgraph of $G'' = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_S)$.*

Notice that the biconnected core is not the 2-core of the original graph. Subgraphs connected to the biconnected core are called *whiskers* by Leskovec et al. [16]. Formally, whiskers are defined as follows:

DEFINITION 3. *A whisker $W = (\mathcal{V}_W, \mathcal{E}_W)$ is a maximal subgraph of G that can be detached from the biconnected core by removing a bridge,*

where a bridge is defined as follows:

DEFINITION 4. *A bridge is a biconnected component of size one which is directly connected to the biconnected core.*

Let $\mathcal{E}_B$ be all the bridges in a graph. Notice that $\mathcal{E}_B \subseteq \mathcal{E}_S$. On the region which is not included in the biconnected core graph G_C, we define the detached graph G_D as follows:

DEFINITION 5. *$G_D = (\mathcal{V}_D, \mathcal{E}_D)$ is a subgraph of G which is induced by $\mathcal{V} \setminus \mathcal{V}_C$.*

Finally, given the original graph $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{V}$ and $\mathcal{E}$ can be decomposed as follows:

PROPOSITION 1. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \mathcal{V}_C \cup \mathcal{V}_D$ and $\mathcal{E} = \mathcal{E}_C \cup \mathcal{E}_D \cup \mathcal{E}_B$.*

PROOF. This follows from the definitions of the biconnected core, bridges, and the detached graph. □

Table 2: Biconnected core and the detached graph

	Biconnected core		Detached graph	
	No. of vertices (%)	No. of edges (%)	No. of components	Size of the LCC (%)
HepPh	9,945 (88.8%)	116,099 (98.7%)	1,123	21 (0.0019%)
AstroPh	16,829 (94.0%)	195,835 (99.4%)	957	23 (0.0013%)
CondMat	19,378 (90.7%)	89,128 (97.6%)	1,669	12 (0.00056%)
DBLP	264,341 (83.4%)	991,125 (94.4%)	43,093	32 (0.00010%)
Flickr	954,672 (47.9%)	20,390,649 (95.1%)	864,628	107 (0.000054%)
Myspace	1,724,184 (82.7%)	45,096,696 (99.2%)	332,596	32 (0.000015%)
LiveJournal	1,650,851 (93.9%)	42,071,541 (99.7%)	101,038	105 (0.000060%)
Amazon	291,449 (87.0%)	862,836 (93.2%)	25,835	250 (0.00075%)

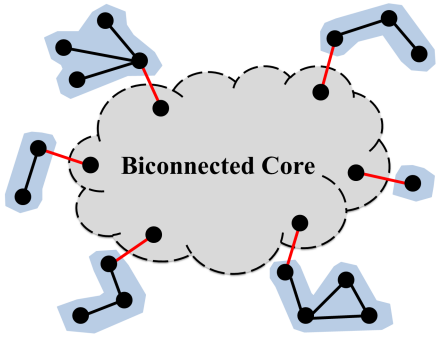


Figure 1: Biconnected core, whiskers, and bridges – grey region indicates the biconnected core where vertices are densely connected to each other, and blue components indicate whiskers. Red edges indicate bridges which connect the biconnected core and each of the whiskers.

Figure 1 illustrates the biconnected core, whiskers, and bridges. The output of our filtering phase is the biconnected core graph where whiskers are filtered out. The filtering phase removes regions of the graph that are clearly partitionable from the remainder. More importantly, there is no overlap between any of the whiskers. This indicates that there is no need to apply overlapping community detection algorithm on the detached regions.

Table 2 shows the sizes of the biconnected core and the connectivity of the detached graph in our real-world networks. Details of these networks are presented in Table 1. We compute the size of the biconnected core in terms of the number of vertices and edges. The number reported in the parenthesis shows how many vertices or edges are included in the biconnected core, i.e., the percentages of $|\mathcal{V}_C|/|\mathcal{V}|$ and $|\mathcal{E}_C|/|\mathcal{E}|$, respectively. We also compute the number of connected components in the detached graph, and the size of the largest connected component (LCC in Table 2) in terms of the number of vertices. The number reported in the parenthesis indicates the relative size of the largest connected component compared to the number of vertices in the original graph.

We can see that the biconnected core contains the substantial portion of the edges. In terms of the vertices, the biconnected core contains around 80 or 90 percentage of the vertices for all datasets except Flickr. In Flickr, the biconnected core only contains around 50 percentage of the vertices while it contains 95 percentage of edges. This indicates that the biconnected core is dense while the detached graph is quite sparse. Recall that the biconnected core is one connected component. On the other hand, in the detached graph, there are many connected components, which implies that the vertices in the detached graph are likely to be disconnected with each other. Notice that each connected component in the detached graph corresponds to a whisker. So, the largest connected component can be interpreted as the largest whisker. Based on the statistics of the detached graph, we can see that whiskers tend to be separable from each other, and there are no significant size whiskers. Also, the size gap between the biconnected core and the largest whisker is significant.

4.2 Seeding Phase

Once we get the biconnected core graph, we find seeds in this filtered graph. The goal of an effective seeding strategy is to identify a diversity of vertices that lie within a cluster of good conductance. This identification should not be too expensive. In this situation, the Andersen-Chung-Lang theorem about the personalized PageRank community finder [4] suggests it is a good method to grow the seeds (Section 4.3).

Graclus centers. One way to achieve these goals is to first apply a high quality and efficient graph partitioning scheme in order to compute a collection of sets with fairly small conductance. For each set (cluster), we find the most central vertex according to the kernel that corresponds to the normalized cut measure. The idea here is roughly that we want something that is close to the partitioning – which ought to be good – but that uses overlap to produce better boundaries between the partitions. See Algorithm 1 for the full procedure. In practice, we perform top-down hierarchical clustering using Graclus [9] to get a large number of clusters. Then, we take the center of each cluster as a seed – the center of a cluster is defined to be the vertex that is closest to the cluster centroid; see step 7 in Algorithm 1. If there are several vertices whose distances are tied for the center of a cluster, we include all of them.

Spread Hubs. From another viewpoint, the goal is to select a set of well-distributed seeds in the graph, such that they will have high coverage after we expand the sets. We greedily choose an independent set of k points in the graph by looking at vertices in order of decreasing degree. For this heuristic, we draw inspiration from the distance function (4), which shows that the distance between a vertex and a cluster is inversely proportional to degree. Thus, high degree vertices are expected to have small distances to many other vertices. This also explains why we call the method *spread hubs*. It also follows from the results in Gleich and Seshadhri [11], which state that there should be good clusters

Algorithm 1 Seeding by Graclus Centers

Input: graph G, the number of seeds k.
Output: the seed set $\mathcal{S}$.

Compute exhaustive and non-overlapping clusters $\mathcal{C}_i$ (i=1, ..., k) on G.
Initialize $\mathcal{S} = \emptyset$.
for each cluster $\mathcal{C}_i$ **do**
  for each vertex $v \in \mathcal{C}_i$ **do**
    Compute $\texttt{dist}(v, \mathcal{C}_i)$ using (4).
  end for
  $\mathcal{S} = \{\operatorname{argmin}_v \texttt{dist}(v, \mathcal{C}_i)\} \cup \mathcal{S}$.
end for

around high degree vertices in power-law graphs with high clustering coefficients. We use an independent set in order to avoid picking seeds nearby each other. Our full procedure is shown in Algorithm 2. As the algorithm proceeds exploring hubs in the network, if there are several vertices whose degrees are the same, we take an independent set of those that are unmarked. This step may result in more than k seeds, however, the final number of returned seeds does not exceed the input k too much because there usually aren't too many high degree vertices.

Algorithm 2 Seeding by Spread Hubs

Input: graph $G = (\mathcal{V}, \mathcal{E})$, the number of seeds k.
Output: the seed set $\mathcal{S}$.

Initialize $\mathcal{S} = \emptyset$.
All vertices in $\mathcal{V}$ are unmarked.
while $|\mathcal{S}| < k$ **do**
  Let $\mathcal{T}$ be the set of unmarked vertices with max degree.
  for each $t \in \mathcal{T}$ **do**
    if t is unmarked **then**
      $\mathcal{S} = \{t\} \cup \mathcal{S}$.
      Mark t and its neighbors.
    end if
  end for
end while

Local Optimal Egonets. This strategy was presented in [11]. Let $\texttt{ego}(s)$ denote the egonet of vertex s which is defined to be the union of s and its neighbors. [11] takes an egonet whose conductance is smaller than the conductance of any of its neighbors' egonets, that is, they select a seed s such that

$$\texttt{cond}(\texttt{ego}(s)) \leq \texttt{cond}(\texttt{ego}(v))$$

for all v adjacent to s.

Random Seeds. Given the number of seeds k, we randomly take k seeds in the graph. Andersen and Lang gave some theoretical justification for why this method should be competitive [5].

4.3 Seed Set Expansion Phase

Once we have a set of seed vertices, we wish to expand the clusters around those seeds. An effective technique for this task is a personalized PageRank vector, also known as a random-walk with restart (RWR) [21]. A personalized PageRank vector is the stationary distribution of a random walk that, with probability α follows a step of a random walk and with probability $(1-\alpha)$ jumps back to a seed node. If there are multiple seed nodes, then the choice is usually uniformly random. Thus, nodes close by the seed are more likely to be visited. Recently, such techniques were shown to produce communities that best match communities found in real-world networks [2]. In fact, personalized PageRank vectors have surprising relationships to graph cuts and clustering methods [4]. Andersen et al. show that a particular algorithm to compute a personalized PageRank vector, followed by a sweep over all cuts induced by the vector, will identify a set of good conductance within the graph. They proved this via a "localized Cheeger inequality" that states, informally, that the set identified via this procedure has a conductance that isn't too far away from the best conductance of any set containing that vertex. More recently, Mahoney et al. [17] show that personalized PageRank is, effectively, a seed-biased eigenvector of the Laplacian. They also show a limit to relate the personalized PageRank vectors to the Fiedler vector of a graph.

Algorithm 3 Find a low conductance set near a seed

Input: graph G, seed set $\mathcal{S}$, PageRank link-following probability parameter $0 < \alpha < 1$, accuracy $\varepsilon > 0$
Output: low conductance set $\mathcal{C}$.

Initialize $x_v, r_v = 0$ for $v \in \mathcal{V}$, set $r_v = 1/|\mathcal{S}|$ for all $v \in \mathcal{S}$
while Any $r_v > \texttt{deg}(v)\varepsilon$, set v to this vertex **do**
  Update $x_v = x_v + (1-\alpha)r_v$.
  For each $(v, u) \in E$,
    update $r_u = r_u + \alpha r_v / (2\,\texttt{deg}(u))$
  Update $r_v = \alpha r_v / 2$
end while
Sort vertices by decreasing $x_v / \texttt{deg}(v)$
For each prefix set of vertices in the sorted list, compute the conductance of that set and set $\mathcal{C}$ to be the set that achieves the minimum.

We briefly summarize the procedure in Algorithm 3. Please see Andersen et al. [4] for a full description of the algorithm. This procedure is closely related to a coordinate descent optimization procedure [6] on the PageRank linear system. Although it may not be apparent from the procedure, this algorithm is remarkably efficient when combined with appropriate data structures. The algorithm keeps two vectors of values for each vertex, $\mathbf{x}$ and $\mathbf{r}$. In a large graph, most of these values will remain zero on the vertices and hence, these need not be stored. Our implementation uses a hash table for the vectors $\mathbf{x}$ and $\mathbf{r}$. Consequently, the sorting step is only over a small fraction of the total vertices. Typically, we find this method takes only a few milliseconds, even for a large graph.

In the personalized PageRank clustering scheme, there are two parameters: α and ε. We follow standard practice for PageRank clustering on an undirected graph and set $\alpha = 0.99$ [16]. This value yields results that are similar to those without damping, yet have bounded computational time. The parameter ε is an accuracy parameter. As $\varepsilon \to 0$, the final vector solution $\mathbf{x}$ tends to the exact solution of the PageRank linear system. When used for clustering, however, this parameter controls the effective *size* of the final cluster. If ε is large (about 10^{-2}), then the output vector is inaccurate, incredibly sparse, and the resulting cluster is small. If ε is small, say 10^{-8}, then the PageRank vector is accurate, nearly dense, and the resulting cluster may be large. We thus run the PageRank clustering scheme about

13 times, with a range of accuracy parameters that are empirically designed to produce clusters with between 1 and 50,000 times the number of edges in the initial seed set. The final community we select is the one with the best conductance score from these 13 possibilities. The seed set for each run is the vertex neighborhood of a seed node. As reported in [11], we found that this yielded better performance and larger clusters.

4.4 Propagation Phase

Once we get the personalized PageRank communities on the biconnected core graph, we further expand each of the communities to the regions that we detached in the filtering phase. Our assignment procedure is straightforward: for each detached whisker connected via a bridge, we add that piece to all of the clusters that utilize the other vertex in the bridge. This procedure is described in Algorithm 4. In this way, each community $\mathcal{C}_i$ is expanded.

We now show that this method only improves the final clustering result in terms of the normalized cut metric. To do this, we need to fix some notation. Let $\mathcal{E}_{B_i}$ be a set of bridges which are attached to $\mathcal{C}_i$, and $W_{\mathcal{C}_i}$ be a set of whiskers which are attached to the bridges, i.e., $W_{\mathcal{C}_i} = (\mathcal{V}_{W_i}, \mathcal{E}_{W_i})$ where

$$w_j = (\mathcal{V}_j, \mathcal{E}_j) \in W_{\mathcal{C}_i};\ \ \mathcal{V}_{W_i} = \bigcup_{w_j \in W_{\mathcal{C}_i}} \mathcal{V}_j;\ \text{ and }\ \mathcal{E}_{W_i} = \bigcup_{w_j \in W_{\mathcal{C}_i}} \mathcal{E}_j.$$

Finally, let $\mathcal{C}_i'$ denote the expanded $\mathcal{C}_i$, where $|\mathcal{C}_i'| \geq |\mathcal{C}_i|$. Equality holds in this expression when there is no bridge attached to $\mathcal{C}_i$. When we expand $\mathcal{C}_i$ using Algorithm 4, $\mathcal{C}_i'$ is equal to $\{\mathcal{C}_i \bigcup \mathcal{V}_{W_i}\}$. The following results show that we only decrease the size of the (normalized) cut by adding the whiskers.

THEOREM 1. *If a community $\mathcal{C}_i$ is expanded to $\mathcal{C}_i'$ using Algorithm 4, $\texttt{cut}(\mathcal{C}_i') = \texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)$.*

PROOF. Recall that $\texttt{cut}(\mathcal{C}_i)$ is defined as follows:

$$\begin{aligned}\texttt{cut}(\mathcal{C}_i) &= \texttt{links}(\mathcal{C}_i, \mathcal{V} \setminus \mathcal{C}_i).\\ &= \texttt{links}(\mathcal{C}_i, \mathcal{V}) - \texttt{links}(\mathcal{C}_i, \mathcal{C}_i).\end{aligned}$$

Let us first consider $\texttt{links}(\mathcal{C}_i', \mathcal{V})$ as follows:

$$\texttt{links}(\mathcal{C}_i', \mathcal{V}) = \texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i).$$

Notice that $\texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}) = \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)$. Thus, $\texttt{links}(\mathcal{C}_i', \mathcal{V})$ can be expressed as follows:

$$\texttt{links}(\mathcal{C}_i', \mathcal{V}) = \texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}).$$

Now, let us compute $\texttt{cut}(\mathcal{C}_i')$ as follows:

$$\begin{aligned}\texttt{cut}(\mathcal{C}_i') &= \texttt{links}(\mathcal{C}_i', \mathcal{V}) - \texttt{links}(\mathcal{C}_i', \mathcal{C}_i').\\ &= \texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}) - \texttt{links}(\mathcal{C}_i', \mathcal{C}_i').\end{aligned}$$

Notice that $\texttt{links}(\mathcal{C}_i', \mathcal{C}_i') = \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}) + \texttt{links}(\mathcal{C}_i, \mathcal{C}_i) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)$.
Finally, $\texttt{cut}(\mathcal{C}_i')$ can be expressed as follows:

$$\texttt{cut}(\mathcal{C}_i') = \texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i).$$

□

THEOREM 2. *If a community $\mathcal{C}_i$ is expanded to $\mathcal{C}_i'$ using Algorithm 4, $\texttt{ncut}(\mathcal{C}_i') \leq \texttt{ncut}(\mathcal{C}_i)$.*

PROOF. Recall that

$$\texttt{ncut}(\mathcal{C}_i) = \frac{\texttt{cut}(\mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}.$$

On the other hand, by Theorem 1, we can represent $\texttt{ncut}(\mathcal{C}_i')$ as follows:

$$\begin{aligned}\texttt{ncut}(\mathcal{C}_i') &= \frac{\texttt{cut}(\mathcal{C}_i')}{\texttt{links}(\mathcal{C}_i', \mathcal{V})}.\\ &= \frac{\texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i})}.\end{aligned}$$

Therefore, $\texttt{ncut}(\mathcal{C}_i') \leq \texttt{ncut}(\mathcal{C}_i)$. The equality holds when there is no bridge attached to $\mathcal{C}_i$, i.e., $\mathcal{E}_{B_i} = \emptyset$. □

Algorithm 4 Propagation Module

Input: graph $G = (\mathcal{V}, \mathcal{E})$, biconnected core $G_C = (\mathcal{V}_C, \mathcal{E}_C)$, communities of $G_C : \mathcal{C}_i\ (i = 1, ..., k) \in \mathcal{C}$.
Output: communities of G.

```
for each C_i ∈ C do
    Detect bridges E_{B_i} attached to C_i.
    for each b_j ∈ E_{B_i} do
        Detect the whisker w_j = (V_j, E_j) which is attached to b_j.
        C_i = C_i ∪ V_j.
    end for
end for
```

4.5 Time Complexity Analysis

We summarize the time complexity of our overall algorithm in Table 3. The filtering phase requires computing biconnected components in a graph, which takes $O(|\mathcal{V}| + |\mathcal{E}|)$ time. The complexity of "Graclus centers" seeding strategy is determined by the complexity of hierarchical clustering using Graclus. Recall that "Spread hubs" seeding strategy requires nodes to be sorted according to their degrees. Thus, the complexity of this strategy is bounded by the sorting operation. Let $\texttt{deg}_{max}$ denote the maximum degree in G_C. The "Local optimal egonet" seeding requires computing triangles, which takes $O(|\mathcal{E}_C|\,\texttt{deg}_{max})$. Expanding each seed requires solving multiple personalized PageRank clustering problems. The complexity of this operation is complicated to state compactly [4], but it scales with the output size of the final cluster, $\texttt{links}(\mathcal{C}_i, \mathcal{V}_C)$. Finally, our simple propagation procedure scans the regions that were not included in the biconnected core and attaches them to a cluster.

Table 3: Time complexity of each phase.

Phase		Time complexity
Filtering		$O(\lvert\mathcal{V}\rvert + \lvert\mathcal{E}\rvert)$
Seeding	Graclus centers	$O(\lceil \log k \rceil (\lvert\mathcal{V}_C\rvert + \lvert\mathcal{E}_C\rvert))$
	Spread hubs	$O(\lvert\mathcal{V}_C\rvert \log \lvert\mathcal{V}_C\rvert + k)$
	Local egonets	$O(\lvert\mathcal{E}_C\rvert\,\texttt{deg}_{max})$
	Random	$O(k)$
Seed expansion		$O(\sum_i^k \texttt{links}(\mathcal{C}_i, \mathcal{V}_C))$
Propagation		$O(\sum_i^k (\mathcal{E}_{B_i} + \mathcal{V}_{W_i} + \mathcal{E}_{W_i}))$

5. EXPERIMENTAL RESULTS

We compare our seed set expansion algorithm with two other state-of-the-art overlapping community detection methods: Demon [8] and Bigclam [27]. For Demon, and Bigclam, we used the software which is provided by the authors of [8] and [27], respectively. All the experiments are performed on a computer with a Xeon X5440 2.83GHz CPU and 32GB memory. In Demon, we set $\epsilon = 0.3$. In Bigclam, we use default parameter settings the software provides. Our seed

set expansion algorithm is written in a mixture of C++ and MATLAB. Bigclam supports multi-threaded execution.

In the following experiments: the labels "demon" and "bigclam" refer to the output from these methods. We refer to our method by the origin of the seeding strategy discussed in Section 4.2. Both "graclus centers" and "spread hubs" are the new methods we propose in this manuscript. The "egonet" method uses the seeding strategy from Gleich and Seshadhri [11], and "random" refers to random seeds.

5.1 Graph coverage

In the first experiment we conduct, we report on the output of each of the six methods on the eight networks which are presented in Table 1. Table 4 shows the returned number of clusters and the graph coverage of each algorithm. The "demon" implementation fails on Flickr, Myspace and LiveJournal on a computer with 32GB memory. (In fact, we tried this algorithm on another machine with 256GB memory and it also failed.) Using 20 threads, "bigclam" does not finish on the Myspace network after running for one week.

The graph coverage indicates how many vertices are assigned to clusters (i.e., the number of assigned vertices divided by the total number of vertices in a graph). Note that we can control the number of seeds k in "graclus centers", "spread hubs", and "random" seeds. We also can specify the number of clusters k in "bigclam". We set k (in our methods and "bigclam") as 100 for HepPh, 200 for AstroPh and CondMat, 15,000 for Flickr, Myspace, and LiveJournal, and 25,000 for DBLP and Amazon. Since we remove duplicate clusters after the PageRank expansion, the returned number of clusters can be smaller than k. Also, since we choose all the tied seeds in "graclus centers" and "spread hubs", the returned number of clusters of these algorithms can be slightly larger than k. Recall that we use a top-down hierarchical clustering scheme in the "graclus centers" strategy. So, in this case, the returned number of clusters before filtering the duplicate clusters is slightly greater than or equal to $2^{\lceil \log k \rceil}$. On the other hand, the number of seeds in "egonets" is determined within the seeding algorithm. Demon also determines the number of clusters based on datasets. We can see that "graclus centers" and "spread hubs" methods cover larger portions of the graph than other methods across all the datasets.

5.2 Community quality using conductance

We evaluate the quality of overlapping clustering in terms of the maximum conductance of any cluster. A high quality algorithm should return a set of clusters that covers a large portion of the graph with small maximum conductance. This objective function has been studied theoretically in the context of overlapping clustering [13] and there exists an approximation algorithm, although it is expensive computationally.

Figure 2 shows the quality-vs-coverage for the six algorithms we study on the six networks where we do not have ground truth community information. For each method, we first sort the clusters according to conductance measure in ascending order, and then greedily take clusters until a certain percentage of the graph is covered. The x-axis of each plot is the graph coverage, and the y-axis is the maximum conductance value among the clusters we take. We can interpret this plot as follows: we need to use the cluster whose conductance score is y to cover x percentage of the graph. Note that lower conductance indicates better quality of clusters. That is, the lower curve indicates better clusters. As can be seen in the plots, our algorithm with "graclus centers" seeding strategy outperforms the other methods. Also the simple "spread hubs" outperforms "egonet", random seeds, Demon, and Bigclam. The original motivation for our studies in this paper was that egonet seeding did not produce high coverage.

5.3 Community quality via ground truth

We have ground truth communities for the DBLP and Amazon networks, thus, for these networks, we compare against these communities instead of using the conductance measure. In DBLP, each publication venue (i.e., journal or conference) corresponds to an individual ground truth community. In the Amazon network, each ground truth community is defined to be a product category that Amazon provides. Given a set of algorithmic communities C and the ground truth communities S, we compute F_1 measure and F_2 measure to evaluate the relevance between the algorithmic communities and the ground truth communities. In general, F_β measure is defined as follows:

$$F_\beta(S_i) = (1+\beta^2)\frac{precision(S_i)\cdot recall(S_i)}{\beta^2 \cdot precision(S_i) + recall(S_i)}$$

where β is a non-negative real value, and the *precison* and *recall* of $S_i \in S$ are defined as follows:

$$precision(S_i) = \frac{|C_j \bigcap S_i|}{|C_j|},$$

$$recall(S_i) = \frac{|C_j \bigcap S_i|}{|S_i|},$$

where $C_j \in C$, and $F_\beta(S_i) = F_\beta(S_i, C_{j^*})$ where $j^* = \underset{j}{\text{argmax}}$ $F_\beta(S_i, C_j)$. Then, the average F_β measure is defined to be

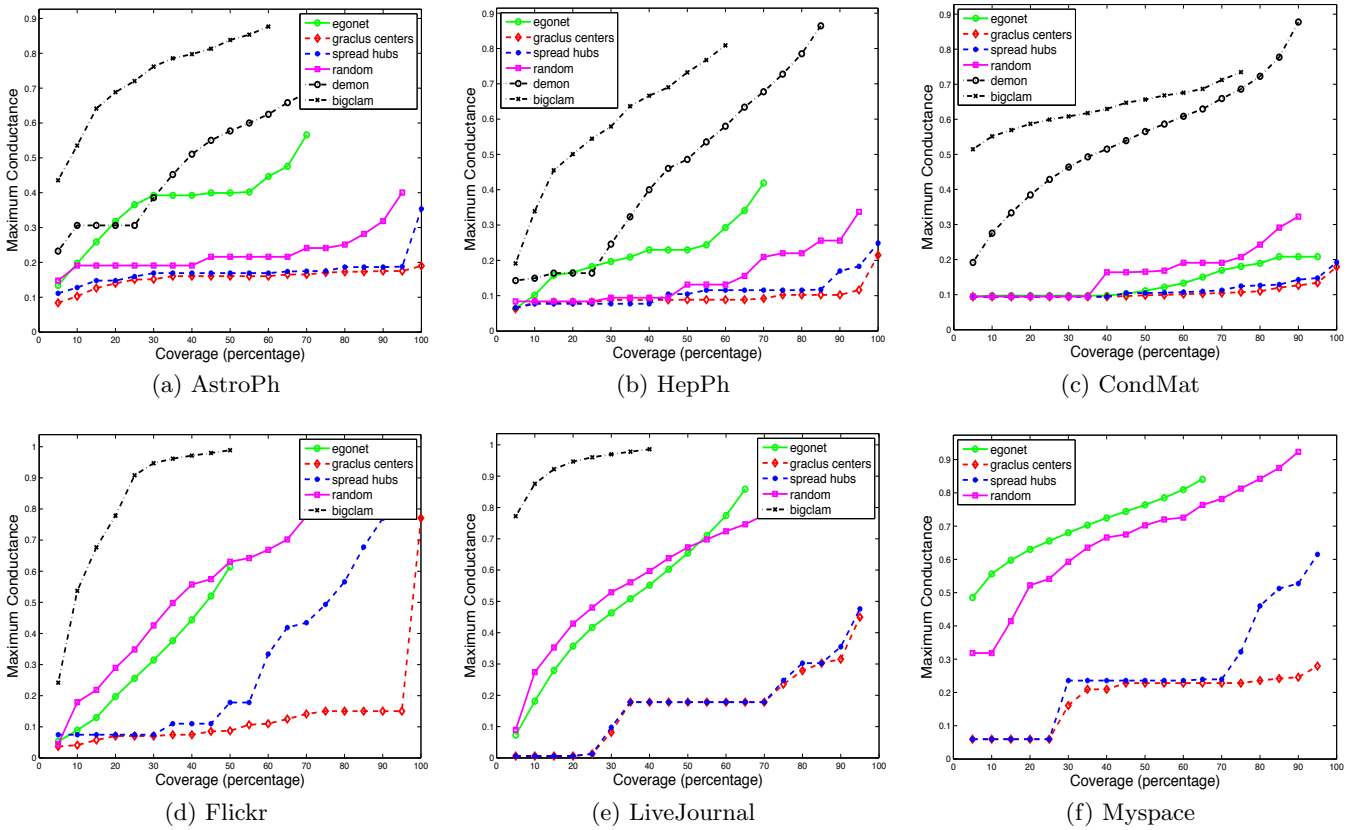
$$\bar{F_\beta} = \frac{1}{|S|}\sum_{S_i \in S} F_\beta(S_i).$$

Given an algorithmic community, *precision* indicates how many vertices are actually in the same ground truth community. Given a ground truth community, *recall* indicates how many vertices are predicted to be in the same community in a retrieved community. By definition, the precision and the recall are evenly weighted in F_1 measure. On the other hand, the F_2 measure puts more emphasis on recall than precision. The authors in [27] who provided the datasets argue that it is important to quantify the recall since the ground truth communities in these datasets are partially annotated, i.e., some vertices are not annotated to be a part of the ground truth community even though they actually belong to that community. This indicates that it would be reasonable to weight recall higher than precision, which is done by the F_2 measure.

In Figure 3, we report the average F_1 and F_2 measures on DBLP and Amazon networks. On the DBLP network, "spread hubs" is the best, and "graclus centers" is the second best in terms of F_1 measure. With respect to F_2 measure, "graclus centers" is the best and "spread hubs" is the second best. On Amazon network, "spread hubs" is the best in terms of both F_1 and F_2 measures. We also notice that all seed set expansion algorithms outperform Demon and Bigclam – even when we use "random" seeding strategy. We discuss this observation more comprehensively in our discussion section.

Table 4: Returned number of clusters and graph coverage of each algorithm

Graph		random	egonet	graclus ctr.	spread hubs	demon	bigclam
HepPh	coverage (%)	97.1	72.1	100	100	88.8	62.1
	no. of clusters	97	241	109	100	5,138	100
AstroPh	coverage (%)	97.6	71.1	100	100	94.2	62.3
	no. of clusters	192	282	256	212	8,282	200
CondMat	coverage (%)	92.4	99.5	100	100	91.2	79.5
	no. of clusters	199	687	257	202	10,547	200
DBLP	coverage (%)	99.9	86.3	100	100	84.9	94.6
	no. of clusters	21,272	8,643	18,477	26,503	174,627	25000
Amazon	coverage (%)	99.9	100	100	100	79.2	99.2
	no. of clusters	21,553	14,919	20,036	27,763	105,828	25,000
Flickr	coverage (%)	76.0	54.0	100	93.6	-	52.1
	no. of clusters	14,638	24,150	16,347	15,349	-	15,000
LiveJournal	coverage (%)	88.9	66.7	99.8	99.8	-	43.9
	no. of clusters	14,850	34,389	16,271	15,058	-	15,000
Myspace	coverage (%)	91.4	69.1	100	99.9	-	-
	no. of clusters	14,909	67,126	16,366	15,324	-	-



(a) AstroPh (b) HepPh (c) CondMat

(d) Flickr (e) LiveJournal (f) Myspace

Figure 2: Conductance vs. graph coverage – lower curve indicates better communities. Overall, "graclus centers" outperforms other seeding strategies, including the state-of-the-art methods Demon and Bigclam.

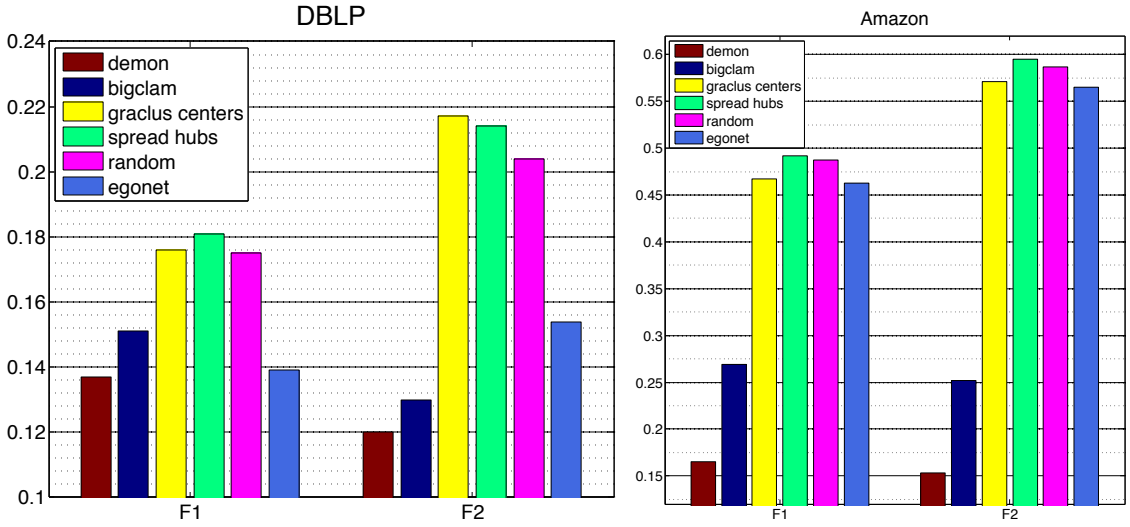


Figure 3: F1 and F2 measures comparing our algorithmic communities to ground truth – a higher bar indicates better communities.

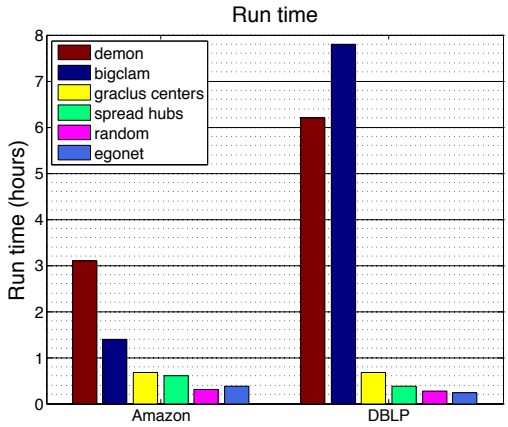


Figure 4: Runtime on Amazon and DBLP – The seed set expansion algorithm is faster than Demon and Bigclam.

5.4 Comparison of running times

Finally, we compare the algorithms by runtime. Figure 4 and Table 5 show the runtime of each algorithm. We run the single thread version of Bigclam for HepPh, AstroPh, CondMat, DBLP, and Amazon networks, and use the multi-threaded version with 20 threads for Flickr, Myspace, and LiveJournal networks.

As can be seen in Figure 4, the seed set expansion methods are much faster than Demon and Bigclam on DBLP and Amazon networks. On small networks (HepPh, AstroPh, CondMat), our algorithm with "spread hubs" is faster than Demon and Bigclam. On large networks (Flickr, LiveJournal, Myspace), our seed set expansion methods are much faster than Bigclam even though we compare a single-threaded implementation of our method with 20 threads for Bigclam.

6. DISCUSSION AND CONCLUSION

We now discuss the results from our experimental investigations. First, we note that our seed set expansion method was the only method that worked on all of the problems. Also, our method is faster than both Bigclam and Demon. Our seed set expansion algorithm is also easy to parallelize because each seed can be expanded independently. This property indicates that the runtime of the seed set expansion method could be further reduced in a multi-threaded version. Also, we can use any other high quality partitioning scheme instead of Graclus including those with parallel and distributed implementations [25]. Perhaps surprisingly, the major difference in cost between using Graclus centers for the seeds and the other seed choices does not result from the expense of running Graclus. Rather, it arises because the personalized PageRank expansion technique takes longer for the seeds chosen by Graclus and spread hubs. When the PageRank expansion method has a larger input set, it tends to take longer, and the input sets we provide for the spread hubs and Graclus seeding strategies are the neighborhood sets of high degree vertices.

Another finding that emerges from our results is that using random seeds outperforms both Bigclam and Demon. We believe there are two reasons for this finding. First, random seeds are likely to be in *some* set of reasonable conductance as also discussed by Andersen and Lang [5]. Second, and importantly, a recent study by Abrahao [2] showed that personalized PageRank clusters are topologically similar to real-world clusters [2]. Any method that uses this technique will find clusters that look real.

Finally, we wish to address the relationship between our results and some prior observations on overlapping communities. The authors of Bigclam found that the dense regions of a graph reflect areas of overlap between overlapping communities. By using a conductance measure, we ought to find only these dense regions – however, our method produces much larger communities that cover the entire graph. The reason for this difference is that we use the entire vertex neighborhood as the restart for the personalized PageRank expansion routine. We avoid seeding exclusively inside a dense region by using an entire vertex neighborhood as a seed, which grows the set beyond the dense region. Thus, the communities we find likely capture a combination of communities given by the egonet of the original seed node. To expand on this point, in experiments we omit due to space, we found that seeding solely on the node itself – rather than us-

Table 5: Running times of different methods on our test networks

Graph	random	egonet	spread hubs	graclus ctr.	bigclam	demon
HepPh	21s	25s	31s	4m	12m	46s
AstroPh	10s	17s	41s	3m 30s	49m	1m
CondMat	7s	35s	52s	1m 35s	8m	1m 14s
Flickr	30m	40m	1h 40m	6h 49m	59h 35m	-
LiveJournal	34m	56m	2h 18m	4h 12m	50h	-
Myspace	27m	2h 29m	7h 9m	17h 15m	> 7 days	-

ing the vertex neighborhood – resulted in significantly worse performance in terms of conductance-vs-coverage.

Overall, our seed set expansion strategies significantly outperformed both Demon and Bigclam, two state of the art overlapping community methods in runtime, conductance-vs-coverage, and ground-truth accuracy.

Acknowledgments

This research was supported by NSF grants CCF-1117055 and CCF-0916309 to ID, and by NSF CAREER award CCF-1149756 to DG.

7. REFERENCES

[1] Stanford Network Analysis Project. `http://snap.stanford.edu/`.

[2] B. Abrahao, S. Soundarajan, J. Hopcroft, and R. Kleinberg. On the separability of structural classes of communities. In *KDD*, pages 624–632, 2012.

[3] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466:761–764, 2010.

[4] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, 2006.

[5] R. Andersen and K. J. Lang. Communities from seed sets. In *WWW*, pages 223–232, 2006.

[6] F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V. Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and Katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[7] R. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, 1995.

[8] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Demon: a local-first discovery method for overlapping communities. In *KDD*, 2012.

[9] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *PAMI*, 29(11):1944–1957, 2007.

[10] U. Gargi, W. Lu, V. Mirrokni, and S. Yoon. Large-scale community detection on YouTube for topic discovery and exploration. In *ICWSM*, 2011.

[11] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, pages 597–605, 2012.

[12] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *JPDC*, 48:96–129, 1998.

[13] R. Khandekar, G. Kortsarz, and V. Mirrokni. Advantage of overlapping clusters for minimizing conductance. In *LATIN*, pages 494–505, 2012.

[14] R. I. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *ICML*, 2002.

[15] D. Lai, X. Wu, H. Lu, and C. Nardini. Learning overlapping communities in complex networks via non-negative matrix factorization. *Int. J. Mod Phys C*, 22(10):1173–1190, 2011.

[16] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[17] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *JMLR*, 13:2339–2365, 2012.

[18] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Clustering social networks. In *WAW*, 2007.

[19] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the Flickr social network. In *The First Workshop on Online Social Networks*, 2008.

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

[21] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[22] B. S. Rees and K. B. Gallagher. Overlapping community detection by collective friendship group inference. In *ASONAM*, pages 375–379, 2010.

[23] H. Shen, X. Cheng, K. Cai, and M.-B. Hu. Detect overlapping and hierarchical community structure in networks. *Phys. A*, 388(8):1706–1712, 2009.

[24] H. H. Song, B. Savas, T. W. Cho, V. Dave, I. Dhillon, Y. Zhang, and L. Qiu. Clustered embedding of massive social networks. In *SIGMETRICS*, 2012.

[25] J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *ICDM*, pages 705–714, 2012.

[26] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. *ACM Computing Surveys*, 2013.

[27] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, pages 587–596, 2013.

[28] S. Zhang, R.-S. Wang, and X.-S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Phys. A*, 374(1):483 – 490, 2007.