Local Spectral Clustering for Overlapping Community Detection

YIXUAN LI, Cornell University
KUN HE, Huazhong University of Science and Technology
KYLE KLOSTER, North Carolina State University
DAVID BINDEL and JOHN HOPCROFT, Cornell University

17

Large graphs arise in a number of contexts and understanding their structure and extracting information from them is an important research area. Early algorithms for mining communities have focused on global graph structure, and often run in time proportional to the size of the entire graph. As we explore networks with millions of vertices and find communities of size in the hundreds, it becomes important to shift our attention from macroscopic structure to microscopic structure in large networks. A growing body of work has been adopting local expansion methods in order to identify communities from a few exemplary seed members.

In this article, we propose a novel approach for finding overlapping communities called LEMON (*L*ocal *E*xpansion via *M*inimum *O*ne *N*orm). Provided with a few known *seeds*, the algorithm finds the community by performing a local spectral diffusion. The core idea of LEMON is to use short random walks to approximate an invariant subspace near a seed set, which we refer to as *local spectra*. Local spectra can be viewed as the low-dimensional embedding that captures the nodes' closeness in the local network structure. We show that LEMON's performance in detecting communities is competitive with state-of-the-art methods. Moreover, the running time scales with the size of the community rather than that of the entire graph. The algorithm is easy to implement and is highly parallelizable. We further provide theoretical analysis of the local spectral properties, bounding the measure of tightness of extracted community using the eigenvalues of graph Laplacian.

We thoroughly evaluate our approach using both synthetic and real-world datasets across different domains, and analyze the empirical variations when applying our method to inherently different networks in practice. In addition, the heuristics on how the seed set quality and quantity would affect the performance are provided.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **World Wide Web**; • **Computing methodologies** → *Machine learning*;

Additional Key Words and Phrases: Community detection, local spectral clustering, seed set expansion, random walk, graph diffusion

This research work was supported by US Army Research Office W911NF-14-1-0477, and National Science Foundation of China 61472147.

Authors' addresses: Y. Li, 350 Gates Hall, Cornell University, Ithaca NY 14853; email: yl2363@cornell.edu; K. He, Department of Computer Science, Huazhong University of Science and Technology, Wuhan, China, 430073; email: brooklet60@hust.edu.cn; K. Kloster, Department of Computer Science, NC State University, 890 Oval Drive, 3280 EBII Raleigh, NC 27606; email: kakloste@ncsu.edu; D. Bindel and J. Hopcroft, 120 Gates Hall, Cornell University, ithaca NY 14853; emails: {bindel, jeh}@cs.cornell.edu.

https://doi.org/10.1145/3106370

ACM Reference format:
Yixuan Li, Kun He, Kyle Kloster, David Bindel, and John Hopcroft. 2018. Local Spectral Clustering for Overlapping Community Detection. *ACM Trans. Knowl. Discov. Data.* 12, 2, Article 17 (January 2018), 27 pages.
https://doi.org/10.1145/3106370

1 INTRODUCTION

Analyzing complex networks to uncover structure and extract information is an important research area. In particular, a significant corpus of literature has studied the tasks of finding structure in networks and identifying communities [9].

In early work, researchers assumed that communities were disjoint and had more internal connections than external connections. Both assumptions have been discarded since in most real-world networks a vertex belongs to more than one community. For instance, in social networks, one might belong to a work community, a community of friends, and a community of individuals that share the same hobby such as golf; in co-purchased networks, one item might belong to multiple categories. Also, since we are dealing with networks with hundreds of millions of vertices, an individual in a community of size 100[1] will certainly have more links outside the community than inside. These key insights have motivated us to identify communities from a new perspective.

A large portion of the community detection literature has focused on the global structure of networks. These globally based detection algorithms usually run in time proportional to the size of the entire graph, which is a major drawback in computational cost. Nowadays, we explore networks with millions of vertices or more to find communities of size one hundred or less. Thus, algorithms whose runtime depends on the size of the entire graph might no longer serve as an ideal solution. It is therefore important to rethink the problem from the perspective of *local structure*, and develop new approaches that enable finding communities in time proportional to the size of the community.

Quite recently, there has been a growing interest in finding communities by locally expanding a small seed set [4, 14, 29, 31]. This type of algorithm usually starts with a few members that are already known to be in the target community, and the goal is to uncover the remaining members in the community as the exemplary members. These known members are usually referred to as *seeds* in the literature, and the process of growing the seed set gradually into a larger set until the target community is revealed is called *seed set expansion* (SSE). The setting of SSE can be widely applied to real-world applications. For example, in web search, with a few known pages that share similar information, we could generate a larger group of web pages that contains the relevant contents with respect to a certain search query; in product networks, SSE enables the automatic categorizing of products that are discovered to be in the same community as the labeled items.

The random walk technique has been extensively adopted as a subroutine for locally growing the seed set in the literature [4, 11, 14, 24, 26, 29, 31]. The dynamics of random walks are effective in finding a local community since they make non-uniform expansion decisions based on the structure revealed during the exploration of the neighborhood surrounding the seeds [4]. This implies that random walk-based local expansion is able to trace the community members in a way that resembles the natural process for forming the local community structure. Recently, through comparing various community detection algorithms, Abrahao et al. found that random walks produce communities that are most structurally similar to real-world communities [1].

In this article, we propose a novel approach, Lemon (Local Expansion via Minimum One Norm), for finding overlapping communities in large networks. We systematically demonstrate

[1]A statistical study on social networks done by Leskovec et al. [18] has shown that real-world communities with high quality are quite small and usually consist of no more than 100 vertices.

Table 1. Statistics for the Real Networks

Domain	Dataset	Vertices	Links	Average membership	Maximum membership	Community size mean
Product	Amazon	334,863	925,872	0.11	49	39
Collaboration	DBLP	317,080	1,049,866	0.22	11	251
Social	YouTube	1,134,890	2,987,624	0.05	41	79
Social	Orkut	3,072,441	117,185,083	9.56	504	83

that LEMON can achieve both high efficiency and effectiveness that outperforms state-of-the-art proposals by a large margin.

Comparing to traditional spectral clustering methods, LEMON has two significant advantages. First, our local spectral method does not require the burdensome computation of a large number of singular vectors. Instead we use short random walks to approximate an invariant subspace near a seed set, which we refer to as *local spectra*. Local spectra can be viewed as the low-dimensional embedding space that captures the nodes' closeness in the local network structure. Second, traditional spectral methods usually partition the vertices into disjoint communities, whereas our method is able to detect overlapping communities.

We aim to develop a comprehensive understanding of the local spectral approach for identifying a community from a small seed set. Following the central idea of our approach, we seek to solve fundamentally important questions such as what defines "good" communities and when do they emerge as we expand the seed set (Section 4.5)? How can we find a small community in time proportional to the size of the community rather than that of the entire graph (Section 4.4)? What defines "good" seeds and how many seeds could uniquely define a community (Section 5)? And given that networks are not all similar in nature, how is the local expansion approach suited for uncovering communities in different types of networks (Section 6.4)?

We thoroughly evaluate our approach using both synthetic and real-world datasets across different domains, and analyze the empirical variations when applying our method to inherently different networks in practice. We believe that the insights we gained from researching these problems will provide valuable guidance for future investigations of this topic. Code for reproducing our research is publicly available at https://github. com/yixuanli/lemon.

2 RELATED WORK

A considerable amount of literature has been published on finding communities in large social and information networks. We highlight a few ideas that have recently emerged in the literature to clarify how our method differs.

Globally based community finding algorithms. A variety of community detection algorithms have been developed in the past decade, with most of the algorithms falling into the category of global community detection. One category of global algorithms attempts to find communities by optimizing an objective function. For example, GCE [17] identifies maximal cliques as seed communities. It expands these cliques by greedily optimizing a local fitness function. OSLOM [16] also attempts to optimize a fitness function, which expresses the statistical significance of clusters with respect to random fluctuations (i.e., the random graph generated by the configuration model [22] during community expansion). However, the communities identified by mathematical construction may structurally diverge from real communities as pointed out in [1]. Another main stream of research adopts the label propagation approach [25], which defines rules that simulate the spread of labels of vertices in the network. The DEMON algorithm [8], for example, democratically lets each vertex vote for the communities it sees surrounding it in its limited view of the global system using a

label propagation algorithm, and then merges the local communities into a global collection. Other approaches such as Link Community (LC) [2] partitions the graph by first building a hierarchical link dendrogram according to link similarity scores and then cutting the dendrogram at some threshold to yield link communities.

Random walk based detection algorithms. As noted in the preceding section, among the divergent approaches, random walks tend to reveal communities that bear the closest resemblance to the ground truth communities in nature [1]. In the following, we briefly review some methods that have adopted the random walk technique in finding communities. Speaking of methods that focus on the global structure, Pons et al. [24] proposed a hierarchical agglomerative algorithm, *Walk-Trap*, that quantified the similarity between vertices using random walks and then partitioned the network into non-overlapping communities. Meilă et al. [21] presented a clustering approach by viewing the pairwise similarities as edge flows in a random walk and studied the eigenvectors and values of the resulting transition matrix. A later successful algorithm, *Infomap*, proposed by Rosvall and Bergstrom [26] enables uncovering hierarchical structures in networks by compressing a description of a random walker as a proxy for real flow on networks. Variants of this technique such as biased random walk [32] has also been employed in community finding. In [6], Bresson et al. proposed a resampling-based spectral algorithm for multiway graph partitioning.

Local expansion based approaches. To interpret the problem of community detection from a local perspective, our work shares the same spirit as the local expansion algorithms in [4, 12, 14, 29]. Specifically, Andersen and Lang [4] adapted the theoretical results from [28] to expand a set into a community with locally minimal conductance based on lazy random walks. However, the lazy random walk endured a much slower mixing speed and it usually took more than 50,000 steps to converge to a local structure compared with several steps of rapid mixing in a regular random walk. Featuring on the seeding strategies, Whang et al. [29] established several sophisticated methods for choosing the seed set, and then used similar PageRank scheme as that in [3] to expand the seeds until a community with optimal conductance is found. Nonetheless, the performance gained by adopting these intricate seeding methods was not significantly better than that by using random seeds. This implies that a better scheme of expanding the seeds is also needed aside from a good seeding strategy. A recent work by Kloumann and Kleinberg [14] provided a systematic understanding of variants of PageRank-based SSE. They showed many insightful findings regarding the heuristics on seed set. However, the drawback of lacking a proper stop criterion has limited its functionality in practice. The heat kernel algorithm [12] advances PageRank by introducing a different diffusion method.

Local spectra vs. global spectra. Spectral methods is one of the most widely used techniques for exploratory data analysis, with applications including data clustering, image segmentation [5], and community detection, and so on. Spectral clustering makes use of the first few singular vectors of the Laplacian matrix associated with a graph, which are inherently global quantities and may not be sensitive to very local information. For example, in the case when provided with domain knowledge about a target region in the graph, one might be interested in finding clusters *only* near the specified local region in a semi-supervised manner, which might not be otherwise well captured by a method using global eigenvectors. Therefore, in the semi-supervised setting, our pioneer work on local spectral clustering [10, 19][2] have substantial advantage over traditional spectral techniques, with the capability of prioritizing and learning more about a local region of the graph surrounding the seeds. Although the local spectral proposal in [20] incorporates the local information as an additional constraint based on the global spectral methods, the optimization

[2]This manuscript is an extended version of a previous conference publication [19].

Table 2. Symbols and Definitions

Symbol	Definition and description
$\mathcal{S}$	Seed set
C	Detected community
C^*	Ground truth community
$G_{\mathcal{S}}$	Subgraph extracted from the neighborhood surrounding the seed set $\mathcal{S}$
N	Size of the subgraph $G_{\mathcal{S}}$
$\mathbf{A}_{\mathcal{S}}$	Adjacency matrix of subgraph $G_{\mathcal{S}}$
$\bar{\mathbf{A}}_{\mathcal{S}}$	Normalized adjacency matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{D}_{\mathcal{S}}$	Diagonal degree matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{L}_{\mathcal{S}}$	Laplacian matrix of subgraph $G_{\mathcal{S}}$
$\bar{\mathbf{L}}_{\mathcal{S}}$	Normalized Laplacian matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{V}_{k,l}$	l-dimensional local spectral subspace with k-step random walks.
$\Phi(\mathcal{V})$	Conductance of the node set $\mathcal{V}$
$\lambda_i^{(\mathbf{H})}$	The ith smallest eigenvalues of matrix $\mathbf{H}$
$\mathbf{y}$	Probability indicator vector, where larger value indicates a higher probability being in the same community as the seeds

program involves the entire eigenspace, which is less advantageous than using the partial invariant subspace constructed by the *Krylov subspace* in our approach.

3 PRELIMINARIES

3.1 Problem Statement

Given an unweighted graph $G = (\mathcal{V}, \mathcal{E})$ and a set of members $\mathcal{S}$ in the target community C, where $|C| \ll |V|$ and $|\mathcal{S}| \ll |C|$, we are interested in finding the remaining members in C. Generally speaking, we focus on answering *how to accurately find a small community from a seed set in time functional to the size of the community?*

3.2 Symbols and Definitions

Table 2 summarizes a list of the different symbols we will use throughout the article. In general, we use italic letters, e.g., n, to denote scalars; lower boldface characters, e.g., $\mathbf{y}$, to denote vectors; uppercase boldface characters, e.g., $\mathbf{A}$, to denote matrices; and script characters, e.g., C, to denote sets.

3.3 Datasets

3.3.1 Synthetic Datasets. The LFR benchmark graphs [15] have been widely adopted for the purpose of evaluating the performance of community detection algorithms. LFR datasets are generated with built-in community structure that resembles the features found in most real-world networks with power-law degree distribution. It provides researchers with rich flexibility to control the network topology by tuning different parameters, including the graph size n, the average degree $\bar{k}$, the maximum degree k_{max}, the minimum and maximum community size $|C|_{min}$ and $|C|_{max}$, the mixing parameter *mu*, the overlapping membership *om*, and the number of vertices with overlapping membership *on*. Among these parameters, the mixing parameter *mu* has the most significant impact on the network topology, which controls the fraction of links for each vertex that cross to a community with which the vertex is not associated. Usually, larger *mu* would result in lower detection accuracy.

Table 3. Parameters for Generating the LFR Synthetic Datasets

Parameter	Description	Value
n	Graph size	5,000
mu	Mixing parameter	{0.1, 0.3}
$\bar{k}$	Average degree	10
k_{max}	Maximum degree	50
$\|C\|_{min}$	Minimum community size	20
$\|C\|_{max}$	Maximum community size	100
τ_1	Node degree distribution exp.	2
τ_2	Community size distribution exp.	1
om	Maximum number of communities that a node belongs to	{2, 3, . . . , 8}
on	Number of nodes belonging to multiple communities	2,500

Xie et al. [30] have thoroughly compared the performance of different state-of-the-art overlapping community detection algorithms on LFR benchmark datasets. To make the performance evaluation of our algorithm consistent with that in [30], we adopt the same parameters in our article. In total, we generate two sets of networks with mixing parameter $mu = 0.1$ and $mu = 0.3$, respectively. We vary the parameter om from 2 to 8 for each mu and obtain a total of 14 networks. Table 3 lists the value of the parameters we have used for generating the LFR datasets.

3.3.2 Real Datasets. For the purpose of testing on real networks, we include four datasets with ground truth community membership from Stanford Network Analysis Project.[3] These datasets span various domains of network applications, including product networks (Amazon), collaboration networks (DBLP), and online social networks (YouTube and Orkut).[4] Each of the networks can be viewed as an undirected, unweighted, connected graph. The statistical information of the datasets is summarized in Table 1.

3.4 Evaluation Metric

For evaluation, we adopt F1 score to quantify the similarity between the algorithmic community C^* and the ground truth community C. The F1 score for each pair of (C, C^*) is defined by

$$F_1(C, C^*) = \frac{2 \cdot Precision(C, C^*) \cdot Recall(C, C^*)}{Precision(C, C^*) + Recall(C, C^*)}, \tag{1}$$

where the precision and recall are defined by

$$Precision(C, C^*) = \frac{|C \cap C^*|}{|C^*|}, \tag{2}$$

$$Recall(C, C^*) = \frac{|C \cap C^*|}{|C|}. \tag{3}$$

Throughout the article, unless otherwise pointed out, the experimental results on synthetic data for each instance are given by the statistical mean and standard deviation based on 24 test cases[5], and the experimental results on real datasets for each instance are based on 120 test cases. All the ground truth communities for testing are chosen randomly.

[3] http://snap.stanford.edu.
[4] For all the four real datasets, we adopt the top 5,000 communities that possess the highest quality according to [31].
[5] Each local expansion process from a seed set can be viewed as a test case.

4 LOCAL EXPANSION VIA MINIMIZING ONE NORM

4.1 Background

Spectral clustering makes use of a small number of singular vectors proportional to the number of communities in the network. If a graph has thousands of small communities, it is impractical to calculate a number of singular vectors greater than the number of communities. We are experimenting with a fundamentally new technique, which does not require the burdensome computation of a large number of singular vectors. Before explaining our local spectral approach for finding overlapping communities, it is necessary to make clear what we mean by *local spectra*.

In traditional spectral clustering methods, one finds the first few singular vectors of the Laplacian matrix[6] of a graph G with n vertices. Suppose the first d singular vectors are obtained, one can form an $n \times d$ matrix $\mathbf{V}$ as a low-dimensional embedding space that captures nodes' closeness. Then one associates with each vertex a point in this embedding space whose coordinates are given by the entries of the corresponding row in the matrix. Vertices are clustered using some method such as k-means clustering algorithm. This method is not likely to work well if the communities are small and heavily overlapping with each other.

We make two fundamental changes to this method. The first modification is to overcome the drawback of computing the singular vectors. Intuitively, the vertices around the seed members are more likely to be in the target community; thus, a random walk serves as a natural subroutine to reveal these potential members. We start a random walk from several known members in the target community and run for a few steps. The number of random walk steps should be long enough to reach out to the vertices in the target community, but not long enough to spread out to the entire graph. Instead of considering a single probability vector, we consider the span of a few dimensions of vectors after the short random walks and use it as the approximate invariant subspace (*local spectra*), denoted by $\mathbf{V}$. The second is to enable detecting overlapping communities. Now, suppose we wanted to find all the nodes in the same community as node i, it is equivalent to find rows in $\mathbf{V}$ that are nearly identical to that correspond to node i. In other words, we want to find rows in the invariant subspace that point in nearly the same direction as the seed node i. To do so, we look for a sparse vector in the span of $\mathbf{V}$ such that i is in the support. This is equivalent to solve the minimum 0-norm problem

$$\begin{aligned} \min \quad & ||\mathbf{y}||_0 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}_i \geq 1, \end{aligned}$$

where $\mathbf{y}$ can be viewed as the linear combination of basis vectors in $\mathbf{V}$, weighted by the element in an unknown vector $\mathbf{x}$. To put another way, the first constraint means that there is an $\mathbf{x}$ such that $\mathbf{y} = \mathbf{V}\mathbf{x}$. Furthermore, each element in $\mathbf{y}$ should be non-negative.

In general seeking sparse vectors in a given subspace is a hard problem. We will use 1-norm vector $||\mathbf{y}_1||$ as a proxy for the minimum 0-norm vector, and solve the linear programming problem instead

[6] In the literature, several different definitions of graph Laplacian exist. Readers can refer to [23] for more details, which serves as a good introductory paper on spectral clustering.

$$\begin{aligned} \min \quad & ||\mathbf{y}||_1 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}_i \geq 1. \end{aligned}$$

4.2 Algorithm Overview

In the following, we give a formal description of our local spectral approach LEMON for detecting target communities from a small seed set. Given the input of a set of few vertices $\mathcal{S}$ that are already known to be in the target ground truth community C, the goal of our algorithm is to find community C^* such that the F1 measure for scoring the similarity between C and C^* is maximized.

Step 0. Subgraph sampling: In practice, the unknown members in the target community are more likely to be around the seed members, and are usually a few steps away from the seeds. This observation motivates us to reduce the complexity by taking only a portion of the graph into consideration. Ideally, this partial graph should contain as many vertices in the target community as possible, and maintains a small size of the same scale as that of the target community.[7]

To sample the graph, we expand the seed set using random walk. After a few steps of the random walk, vertices with large probability are more likely to be in the target community while vertices with small probability being reached would be treated as redundant ones. If the target community exists for the seed set, then according to [4], this target community would serve as a bottleneck for the probability to be spread out. It is worthwhile noting that other expansion methods such as breadth-first-search (BFS) would entirely ignore the bottleneck defining the community and rapidly mix with the entire graph before a significant fraction of vertices in the community have been reached. In the following, we use $G_{\mathcal{S}} = (\mathcal{V}_{\mathcal{S}}, \mathcal{E}_{\mathcal{S}})$ denote the subgraph extracted from the neighborhood surrounding the seed set $\mathcal{S}$.

Step 1. Generate the local spectra: Consider the subgraph graph $G_{\mathcal{S}}$ extracted from the neighborhood surrounding the seed set $\mathcal{S}$. We define the normalized adjacency matrix $\bar{\mathbf{A}}_{\mathcal{S}}$ of the graph $G_{\mathcal{S}}$ as

$$\bar{\mathbf{A}}_{\mathcal{S}} \stackrel{\text{def}}{=} \mathbf{D}_{\mathcal{S}}{}^{-1/2}(\mathbf{A}_{\mathcal{S}} + \mathbf{I})\mathbf{D}_{\mathcal{S}}{}^{-1/2}, \tag{4}$$

where $\mathbf{A}_{\mathcal{S}}$ and $\mathbf{D}_{\mathcal{S}}$ denote the adjacency matrix and the diagonal degree matrix of $G_{\mathcal{S}}$, respectively. Consider a random walk starting from exemplary vertices in $\mathcal{S}$. Let $\mathbf{p_0}$ denote the initial probability vector where the total probability is evenly distributed among the seed members. We describe how to efficiently construct the local spectra by iteratively transforming the orthonormal basis starting with a *Krylov subspace* defined below.

Definition 1. The order-$l+1$ Krylov matrix generated by the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and vector $\mathbf{p_0}$ is defined by the probability vectors in l successive random walks

$$\mathcal{K}_{l+1}(\mathbf{A}, \mathbf{p_0}) = \left[\mathbf{p_0}, \mathbf{A}\mathbf{p_0}, \ldots, \mathbf{A}^l\mathbf{p_0}\right]. \tag{5}$$

The column vectors of the Krylov matrix can be orthogonalized, and form the basis vectors of the *Krylov subspace* $\mathcal{K}_{l+1}$. In other words, the Krylov subspace is defined by

$$\mathcal{K}_{l+1}(\mathbf{A}, \mathbf{p_0}) = \text{span}\left(\mathbf{p_0}, \mathbf{A}\mathbf{p_0}, \ldots, \mathbf{A}^l\mathbf{p_0}\right). \tag{6}$$

The main idea of Krylov subspace is to approximate the original eigenvector problem of size n by one of dimension $l+1$, typically much smaller than n. Krylov method finds the largest a few

[7] Assume the scale of the target community is known.

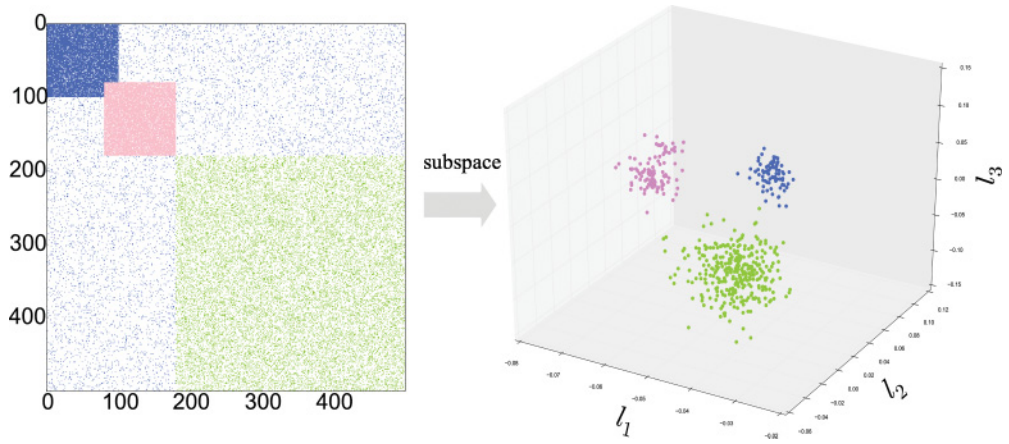

Fig. 1. An example of local spectral subspace $\mathbf{V}_{3,3}$. The synthetic subgraph G_S is generated with Erdős–Rényi $G(n,p)$ model with background noise $p = 0.05$. The clusters A and B (denoted by blue and pink, respectively) are of size 100 with edge probability $p = 0.9$, with partially overlapped 20 nodes. The cluster C (denoted by green) has size 320 with $p = 0.2$. The subspace is generated by Algorithm 1 starting from the seed with index 10 in group A.

eigenvectors of a large matrix in an iterative way [27], which avoids expensive matrix–matrix operations. We may expect the basis vectors of $K_{l+1}(\mathbf{A}, \mathbf{p_0})$ to give good approximations of the eigenvectors corresponding to the $l+1$ largest eigenvalues of $\mathbf{A}$, especially when the dominant eigenvalues decay fast enough.

In Algorithm 1, we briefly summarize the procedure of calculating the local spectral subspace[8] from a specified seed set $\mathcal{S}$. We start by calculating the initial invariant subspace $\mathbf{V}_{0,l}$, which is the orthonormal basis of $\mathcal{K}_{l+1}(\mathbf{A}_\mathcal{S}, \mathbf{p_0})$. And the local spectral subspace can be then obtained by iterating the process specified in Line 4–6 of Algorithm 1. Figure 1 shows an example local spectral subspace $\mathbf{V}_{3,3}$, generated from a synthetic graph with Erdős–Rényi $G(n,p)$ model. In the $G(n,p)$ model, a graph is constructed by connecting n nodes randomly. Each edge is included in the graph with probability p independent from every other edge.

Step 2. Seek for a sparse vector: With the local spectra $\mathbf{V}_{k,l}$, we solve the following linear programming problem,

$$\begin{aligned} \min \quad & ||\mathbf{y}||_1 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}_{k,l}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}(\mathcal{S}) \geq \mathbf{1}, \end{aligned}$$

where the first constraint indicates that $\mathbf{y} \in \mathbb{R}^n$ is in the span of $\mathbf{V}_{k,l}$. Each element in $\mathbf{y}$ indicates the probability for the corresponding vertex to be in the target community, which is non-negative. In other words, the second constraint should be interpreted as element-wise non-negative. The third constraint enforces that seeds are in the support of sparse vector $\mathbf{y}$, where $\mathcal{S}$ is the set of

[8]In the experiments on real datasets, we fix the walk step k and dimension l to be 3 and 3, respectively. For LFR benchmark datasets, we adopt all together six combinations for the (step, dimension) tuple: (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5) and the highest F1 score among these combinations will be returned.

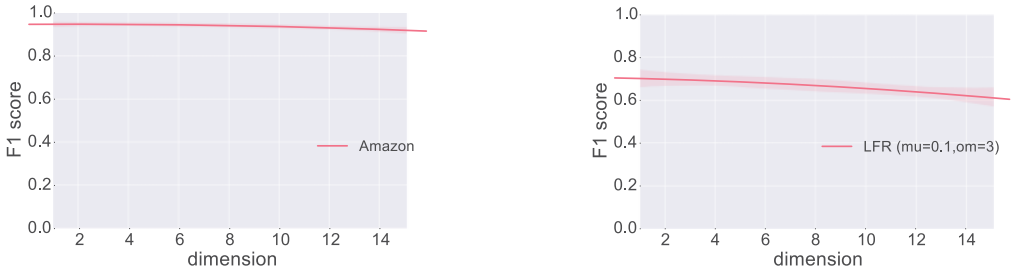

Fig. 2. The average F1 score with varying dimensions l on Amazon dataset (left) and LFR graph (right), respectively. The plots depict the statistical regression line with a 95% confidence interval.

ALGORITHM 1: LocalSpectral($G_{\mathcal{S}}, \mathcal{S}$)

Input: subgraph $G_{\mathcal{S}}$, subspace dimension l, and random walk step k
Output: local spectra $\mathbf{V}_{k,l}$

1: Compute normalized adjacency matrix $\bar{\mathbf{A}}_{\mathcal{S}}$ using (4)
2: Initialize $\mathbf{p}_0$
3: $\mathbf{V}_{0,l} = \mathbf{orth}(K_{l+1}(\bar{A}_{\mathcal{S}}, \mathbf{p}_0))$
4: **for** $i = 1, \ldots, k$ **do**
5: $\quad \mathbf{V}_{i,l} = \mathbf{orth}(\bar{\mathbf{A}}_{\mathcal{S}}\mathbf{V}_{\mathrm{i-1,l}})$
6: **end for**
7: **Return** local spectra $\mathbf{V}_{k,l}$

indices for the seed nodes that we require to be in the community. The entries in the sparse vector $\mathbf{y}$ corresponding to the seeds should be no less than 1.

After sorting the elements in $\mathbf{y}$ in non-ascending order and getting a vector $\hat{\mathbf{y}}$, the vertices corresponding to the top $|C|$ elements in $\hat{\mathbf{y}}$ are returned as the detected community with respect to the seed set $\mathcal{S}$.

Step 3. Reseeding: Augment the initial seed set by merging vertices corresponding to the top $t \leq |C|$ elements of $\hat{\mathbf{y}}$. Denote the augmented seed set as $\mathcal{S}'$. Then repeat step 1 and step 2 using the augmented seed set $\mathcal{S}'$. The detection accuracy can be improved through iterations via increasing t by a constant number s each time. We define s to be the seed expansion step, which is used as a tunable parameter for adjusting the convergence rate. Usually, the larger expansion step would result in lower performance but a faster running speed with less iterations. In the experiments, we fix the seed expansion step to be 6 for both synthetic and real datasets. The number of iterations for the seed expansion is determined by the stop criteria (Section 4.5).

4.3 Parameter Sensitivity

The random walk step k, subspace dimension l, and seed expansion step s are the key parameters in the local spectral clustering algorithm. We conduct parameter sensitivity study for these three parameter on the four real datasets.

4.3.1 Subspace Dimension. To study the effect of subspace dimensionality l, we fix the random walk step to be 3, and vary the number of dimension l from 1 to 15. Figure 2 (left panel) shows that changing the dimension l does not cause significant fluctuation of the performance on Amazon dataset. Choosing a large dimension l is undesirable because it would not only increase

Fig. 3. The average F1 score with varying dimensionality l on Amazon dataset (left) and LFR graph (right), respectively. The plots depict the statistical regression line with a 95% confidence interval.

Fig. 4. The average F1 score with varying seed expansion step s. The plots depict the statistical regression line with a 95% confidence interval.

the computation cost in the step of generating local spectra, but also lead to performance drop as indicated in Figure 2 (right panel). Therefore, choosing a small value is more desirable considering both the computation cost and performance. Throughout the remaining article, we fix $l = 3$ for the real datasets because the experiment suggests that setting $l = 3$ can statistically achieve both high and stable performance.

4.3.2 Random Walk Step. To investigate how the step of random walk affects the algorithm performance, we fix the dimension l to be 3, and vary the random walk step k from 1 to 15. Figure 3 (left panel) shows that the average F1 score plateaus as k increases, and the standard deviation significantly increases when k exceeds 10. This suggests that longer random walk is undesirable for stably uncovering the local community structure. Throughout the remaining article, we fix the random walk step $k = 3$ for the real datasets. For LFR benchmark graphs, we adopt all together six configurations for the (k, l) tuple: $(2, 3)$, $(2, 4)$, $(2, 5)$, $(3, 3)$, $(3, 4)$, $(3, 5)$ and return the highest F1 score among these configurations.

4.3.3 Seed Expansion Step. To study the effect of seed expansion step size s, we fix both the dimension l and random walk step k to be 3, and vary the expansion step s from 2 to 16. Figure 4 shows the results on Amazon (left panel) and LFR (right panel) datasets, respectively. In general, we find our approach is robust to the parameter s when s is kept to a small enough range (e.g., $s \leq 10$). A large incremental step when reseeding is less desirable for stably uncovering the local community structure. Throughout the article, we fix the expansion step $s = 6$ for all experiments.

Table 4. Statistics of the Mean Values for the Sampling Method on Real Datasets

Dataset	Coverage ratio	Sample rate	$\|C\|_{\text{avg}}$	Subgraph size
Amazon	1.00	0.0087	39	2,913
DBLP	0.98	0.0076	251	2,409
YouTube	0.66	0.0033	79	3,745
Orkut	0.64	0.0011	83	3,379

4.4 Complexity Reduction by Sampling Method

In this section, we introduce a sampling method that can effectively solve the memory consumption issue dealing with large graphs.

In the experiments on real datasets, we conduct a random walk starting from the seed set until the probability has been spread out to $\alpha \cdot |C|_{\text{avg}}$ vertices, where α is some constant and $|C|_{\text{avg}}$ is the average community size in the graph.[9] Note that the sample size $\alpha \cdot |C|_{\text{avg}}$ should be large enough to cover as many vertices in the ground truth community as possible, but not too big which otherwise would cover all the nodes in the graph. This newly obtained subgraph will be used as G_S for the remaining computation. The complexity of our algorithm now depends on the size of the subgraph after sampling, which is $O(|C|_{\text{avg}}{}^{\tau})$ for some small constant τ.

Table 4 gives the statistics after applying sampling method to the real networks. For example, in DBLP network, setting α to be around 10 would yield a subgraph containing on average 98% vertices in the ground truth community. After sampling, we only need to deal with a subgraph of size around 2, 400 instead of 317,080, bringing a significant reduction of both temporal and spatial complexity.

4.5 Round Diffusion Vector via Sweeping Cut

If there are ground truth community sizes available, the above algorithm is guaranteed to stop within few iterations since the seed set will no longer augment once its size exceeds that of the ground truth community. The algorithm would then return the community found with the highest F1 score during the iterations as the result. However, in real case, we do not know the exact size of the communities, causing the ambiguity for most locally based detection algorithm to decide when is the proper time to terminate expanding such that the discovered community is a "good" community. It is thus important to solve the two issues: (1) how to automatically determine the size of the community given a seed set $\mathcal{S}$, and (2) when to stop growing the seed set during the reseeding process.

4.5.1 Determine the Size of the Community. It has already been shown that random walks produce communities with conductance guarantees and ensure a small boundary defining a natural community in locally based detection algorithms [4]. The intuition is that adding irrelevant vertices to the target community would inevitably cause the conductance to increase, and finding a low-conductance community could ensure the closeness among the members. A commonly adopted method of rounding the diffusion values into labels is to perform a sweep-cut procedure on the nodes ranked by the diffusion value, with an objective of minimizing the graph cut metric such as *conductance* [3, 20, 29]. As we will see, the local conductance for a small group of vertices in the

[9] A fast implementation method for updating the probability vector of the random walks is featured in detail in [4], Section 4.

graph contains valuable information and enables us to design effective stopping criteria for our algorithm. We define conductance using the generalized Rayleigh quotient specified below:

Definition 2. Let $\mathbf{x} \in \{0,1\}^N$ denote the binary indicator vector for the subset $\tilde{\mathcal{V}} \subseteq \mathcal{V}_s$ and $\mathbf{H} \in \mathbb{R}^{N \times N}$ is any symmetric matrix. The Rayleigh quotient with respect to $\mathbf{H}$ is expressed as the quadratic form of

$$R_{\mathbf{H}}(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{H} \mathbf{x}}{\mathbf{x}^T \mathbf{x}}. \tag{7}$$

In particular, conductance of the set $\tilde{\mathcal{V}}$ measures the fraction of edges leaving $\tilde{\mathcal{V}}$ among all the edges incident on $\tilde{\mathcal{V}}$, and can be expressed using a generalized Rayleigh quotient

$$\Phi(\tilde{\mathcal{V}}) = R_{\mathbf{L}_\mathcal{S}, \mathbf{D}_\mathcal{S}}(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{L}_\mathcal{S} \mathbf{x}}{\mathbf{x}^T \mathbf{D}_\mathcal{S} \mathbf{x}} = \frac{\mathbf{x}^T (\mathbf{D}_\mathcal{S} - \mathbf{A}_\mathcal{S}) \mathbf{x}}{\mathbf{x}^T \mathbf{D}_\mathcal{S} \mathbf{x}}, \tag{8}$$

where $\mathbf{L}_\mathcal{S} = \mathbf{D}_\mathcal{S} - \mathbf{A}_\mathcal{S}$ is the Laplacian matrix of graph $G_\mathcal{S}$.

Now suppose we have a rough estimation of the lower and upper bound for the size of communities in a graph, which we denote by $|C|_{\min}$ and $|C|_{\max}$, respectively. We could modify the original algorithm in the following way.

At step 2, after obtaining the sorted sparse vector $\hat{\mathbf{y}}$, we are hoping to truncate the sorted vector at some point y_g such that all the vertices corresponding to the elements no less than y_g are included in the algorithmic community. The crux lies in that we do not know which is the best position to truncate the vector $\hat{\mathbf{y}}$. To solve this issue, we denote Λ_i as the set of vertices corresponding to the top i elements in $\hat{\mathbf{y}}$. We then sweep over the sets from $\Lambda_{|C|_{\min}}$ to $\Lambda_{|C|_{\max}}$ and calculate the corresponding conductance for each of the sets. In practice, the value of the conductance with respect to varying size would usually change in a non-monotonic pattern that decreases first and then increases later on. We then adopt the minimum conductance encountered on this curve as the estimated size of the community with respect to the seed set $\mathcal{S}$, which we denote by $\Phi_\mathcal{S}^{\min}$.

4.5.2 Stop the Reseeding Process. As we keep augmenting the seed set through reseeding at step 3, a different seed set would result in a different sparse vector $\hat{\mathbf{y}}$ and thus lead to potentially different algorithmic communities. Practically, one of these seed sets during the augmenting process would achieve the highest F1 score. And, it remains to address the issue of when to stop growing the seed set so that it finds the community that resembles most of the ground truth community. This issue can be solved in a similar fashion as that for determining community size. Specifically, we keep track of the value of $\Phi_\mathcal{S}^{\min}$ for different seed set during the expansion, and stop to grow the seed set when $\Phi_\mathcal{S}^{\min}$ reaches a local minimum and starts to increase for the first time.

4.5.3 Auto Detect Size vs. Ground Truth Size. The stop criteria used in LEMON combines methods introduced in Sections 4.5.1 and 4.5.2. To verify the effectiveness, we compare the performance using stop criteria introduced above, with that obtained using ground truth community size $|C|$ explicitly. Figure 5 shows the statistical result of F1 score on both synthetic and real datasets. On both datasets, the F1 score with automatic size determination is only lowered by 10% on average compared with the performance with knowing $|C|$. This implies that our method is applicable for finding the algorithmic community $|C^*|$ that mostly resemble the ground truth community $|C|$ on both synthetic and real datasets in different domains. It also suggests that our method can be applied in practice to uncover natural communities in the situation when no ground truth is available.

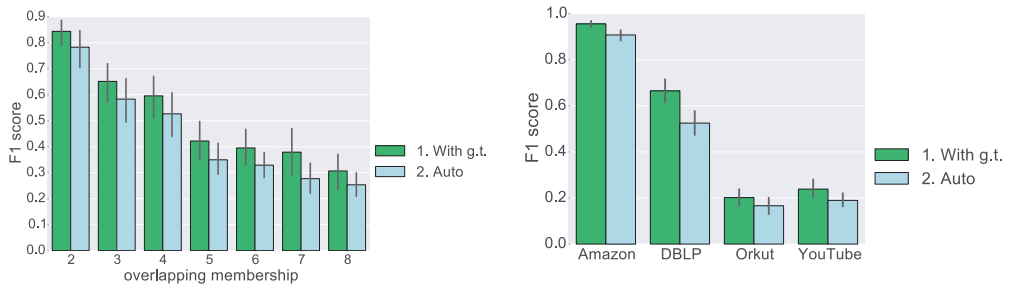

Fig. 5. Comparison of the average F1 score with ground truth and automatic size determination. The left corresponds to the LFR datasets when *mu* = 0.3 and the right corresponds to the real datasets.

4.6 Bounding the Performance

In the following discussion, we bound the measure of "tightness" of the extracted community with respect to the subgraph $G_{\mathcal{S}}$ by relating spectral properties to Rayleigh quotients. We start by providing several theorems and lemmas that will be used for deriving the bound of conductance.

Theorem 1 (Cheeger's Inequality). *Let λ_2 be the second smallest eigenvalue of the Laplacian matrix for a graph $G_{\mathcal{S}}$. Then, $\phi(G_{\mathcal{S}}) \geq \frac{\lambda_2}{2}$, where $\phi(G_{\mathcal{S}}) = \min_{\tilde{\mathcal{V}} \subseteq \mathcal{V}_{\mathcal{S}}} \Phi(\tilde{\mathcal{V}})$.*

There are many proofs known for this theorem [7], and we henceforth omit the details here.

Lemma 1. *The generalized Rayleigh quotient $R_{\mathbf{L}_{\mathcal{S}},\mathbf{D}_{\mathcal{S}}}(\mathbf{x})$ is equivalent to the form of $R_{\bar{\mathbf{L}}_{\mathcal{S}}}(\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x})$, where $\bar{\mathbf{L}}_{\mathcal{S}} = \mathbf{I} - \mathbf{D}_{\mathcal{S}}{}^{-1/2}\mathbf{A}_{\mathcal{S}}\mathbf{D}_{\mathcal{S}}{}^{-1/2}$ is the normalized Laplacian matrix of graph $G_{\mathcal{S}}$.*

Proof. By the definition in Equation (7), we have

$$\begin{aligned} R_{\bar{\mathbf{L}}_{\mathcal{S}}}(\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x}) &= \frac{(\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x})^T \bar{\mathbf{L}}_{\mathcal{S}} (\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x})}{(\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x})^T (\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x})} \\ &= \frac{\mathbf{x}^T \mathbf{D}_{\mathcal{S}}{}^{1/2} \bar{\mathbf{L}}_{\mathcal{S}} \mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \\ &= \frac{\mathbf{x}^T (\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \\ &= R_{\mathbf{L}_{\mathcal{S}},\mathbf{D}_{\mathcal{S}}}(\mathbf{x}). \end{aligned}$$

□

Theorem 2 (Courant–Fischer Theorem). *Let $\mathcal{X}^k$ denote a k dimensional subspace of $\mathbb{R}^N$ and $\mathbf{x} \perp \mathcal{X}^{\mathbf{k}}$ represents that $\mathbf{x} \perp \mathbf{y}$ for all $\mathbf{y} \in \mathcal{X}^{\mathbf{k}}$. For any symmetric matrix $\mathbf{H} \in \mathbb{R}^{\mathbf{N}\times\mathbf{N}}$ with eigenvalues $\lambda_1^{(\mathbf{H})} \leq \lambda_2^{(\mathbf{H})} \leq \cdots \leq \lambda_N^{(\mathbf{H})}$,*

$$\lambda_i^{(\mathbf{H})} = \min_{\mathcal{X}^{N-i-1}} \left(\max_{\mathbf{x} \perp \mathcal{X}^{N-i-1}, \mathbf{x} \neq 0} R_{\mathbf{H}}(\mathbf{x}) \right) = \max_{\mathcal{X}^i} \left(\min_{\mathbf{x} \perp \mathcal{X}^i, \mathbf{x} \neq 0} R_{\mathbf{H}}(\mathbf{x}) \right). \tag{9}$$

We will not include the proof of the Courant–Fischer Theorem here. The interested reader can find a proof in any major linear algebra textbook.

We denote by $\mathbf{D_S}^{1/2}\mathbf{x} = \mathbf{z}$. With the Courant–Fischer Theorem, we can express the eigenvalues of the normalized Laplacian matrix $\bar{\mathbf{L}}_{\mathcal{S}}$ in the following:

$$\begin{aligned}\lambda_i^{(\bar{\mathbf{L}}_{\mathcal{S}})} &= \min_{\mathcal{Z}^{N-i-1}} \left(\max_{\mathbf{z}\perp\mathcal{Z}^{N-i-1}, \mathbf{z}\neq\mathbf{0}} R_{\bar{\mathbf{L}}_{\mathcal{S}}}(\mathbf{z}) \right) \\ &= \min_{\mathcal{Z}^{N-i-1}} \left(\max_{\mathbf{z}\perp\mathcal{Z}^{N-i-1}, \mathbf{z}\neq\mathbf{0}} \frac{\mathbf{x}^T(\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T\mathbf{D}_{\mathcal{S}}\mathbf{x}} \right) \\ &= \min_{\mathcal{X}^{N-i-1}} \left(\max_{\mathbf{x}\perp\mathcal{X}^{N-i-1}, \mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^T(\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T\mathbf{D}_{\mathcal{S}}\mathbf{x}} \right) \\ &= \min_{\mathcal{X}^{N-i-1}} \left(\max_{\mathbf{x}\perp\mathcal{X}^{N-i-1}, \mathbf{x}\neq\mathbf{0}} \frac{\sum_{i\sim j}(x_i - x_j)^2}{\sum_i x_i^2 d_i} \right) \leq 2,\end{aligned}$$

where $i \sim j$ indicates that i is adjacent to j. Similarly,

$$\lambda_i^{(\bar{\mathbf{L}}_{\mathcal{S}})} = \max_{\mathcal{X}^i} \left(\min_{\mathbf{x}\perp\mathcal{X}^i, \mathbf{x}\neq\mathbf{0}} \frac{\sum_{i\sim j}(x_i - x_j)^2}{\sum_i x_i^2 d_i} \right) \leq 2. \tag{10}$$

Corollary 1. *Given a graph $G_{\mathcal{S}}$ with N nodes, the largest eigenvector of its normalized adjacency matrix is no bigger than 2, i.e., $\lambda_N^{(\bar{\mathbf{L}}_{\mathcal{S}})} \leq 2$.*

For any symmetric matrix $\mathbf{H} \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ with orthonormal eigenvectors $\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_N}$, and corresponding eigenvalues $\lambda_1^{(\mathbf{H})} \leq \lambda_2^{(\mathbf{H})} \leq \cdots \leq \lambda_N^{(\mathbf{H})}$, we can always decompose the binary indicator vector $\mathbf{x}$ into a linear combination of the eigenvectors, i.e., $\mathbf{x} = \sum_i a_i\mathbf{q}_i$. This allows us to write

$$R_{\mathbf{H}}(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{H}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \frac{\left(\sum_i a_i\mathbf{q}_i^T\right)\left(\sum_i a_i\lambda_i^{(\mathbf{H})}\mathbf{q}_i\right)}{\left(\sum_i a_i\mathbf{q}_i^T\right)\left(\sum_i a_i\mathbf{q}_i\right)} = \frac{\sum_i a_i^2\lambda_i^{(\mathbf{H})}}{\sum_i a_i^2} = \sum_i w_i\lambda_i^{(\mathbf{H})}, \tag{11}$$

where $w_i = a_i^2/||\mathbf{x}||^2$. Hence, the Rayleigh quotient can be viewed as a weighted average of the eigenvalues. If the indicator vector $\mathbf{x}$ forms an acute angle with the invariant subspace associated with the extreme eigenvalues, then most of the weight in the average must be on eigenvalues close to $\lambda_N^{(\mathbf{H})}$. Similarly, $R_{\mathbf{H}}(\mathbf{x})$ can be bounded from below by the smallest eigenvalues of $\mathbf{H}$, in which case the indicator vector $\mathbf{x}$ can be approximated by a linear combination of the eigenvectors associated with the smallest eigenvalues.

Lemma 2. *Let $\mathbf{x} \in \{0,1\}^N$ denote the binary indicator vector for the algorithmic community $C^* \subseteq \mathcal{V}_s$ corresponding to the seed set $\mathcal{S}$, the conductance of C^* is bounded by*

$$\lambda_2/2 \leq \Phi(C^*) \leq \min\{1, 2(1 - w_1)\}, \tag{12}$$

where λ_2 is the second smallest eigenvalue of Laplacian matrix of $G_{\mathcal{S}}$, and w_1 is the weight of the smallest eigenvalue of the normalized Laplacian matrix $\bar{\mathbf{L}}_{\mathcal{S}}$, as specified in Equation (11).

Proof. The left side inequality holds due to the fact that $\Phi(C^*) \geq \phi(G_{\mathcal{S}})$. Following the Cheeger's inequality that $\phi(G_{\mathcal{S}}) \geq \lambda_2/2$ in Theorem 1, we therefore have $\lambda_2/2 \leq \Phi(C^*)$. To prove the right side, we first express the conductance $\Phi(C^*)$ using Rayleigh quotient,

$$\Phi(C^*) = R_{\mathbf{L}_{\mathcal{S}},\mathbf{D}_{\mathcal{S}}}(\mathbf{x}), \tag{13}$$

which can be further rewritten as $R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x})$, according to Lemma 1. Using similar decomposition as that in Equation (11), we can express $R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x})$ as the weighted average of the eigenvalues $\lambda_1^{(\bar{\mathbf{L}}_S)} \leq \lambda_2^{(\bar{\mathbf{L}}_S)} \leq \cdots \leq \lambda_N^{(\bar{\mathbf{L}}_S)}$, i.e.,

$$\begin{aligned} R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x}) &= \sum_i w_i \lambda_i^{(\bar{\mathbf{L}}_S)} \\ &\leq w_1 \lambda_1^{(\bar{\mathbf{L}}_S)} + (1 - w_1)\lambda_N^{(\bar{\mathbf{L}}_S)} \\ &\leq 2(1 - w_1). \end{aligned}$$

□

5 SEEDING

Since the initial seed set serves as a key component in our algorithm for uncovering the target community C, it is thus crucial to consider how the quality of seed set affect the performance. In practice, there is not much control over how the seeds are selected. However, the alternative seeding methods can be strategically applied by domain experts in different scenarios based on the availability of candidate seeds. In this section, we will focus on addressing two fundamentally important issues regarding the seed set: (1) What defines "good" seeds? and (2) How many seeds are needed in order to uniquely define a community?

5.1 Seeding Method

To give a well-rounded evaluation on this, we encompass in total five different seeding methods here. In this experiment, we adopt $|\mathcal{S}| = 3$ seeds for each of the seeding method listed below.

(1) *High degree seeding*: pick $|\mathcal{S}|$ vertices with degree ranked in the top one third among the degree of all vertices in C.
(2) *Low degree seeding*: pick $|\mathcal{S}|$ vertices with degree ranked in the bottom one third among the degree of all vertices in C.
(3) *Triangle seeding*: pick $|\mathcal{S}|$ vertices in C that form a triangle as the initial seed set.
(4) *Random seeding*: pick $|\mathcal{S}|$ vertices in C randomly.
(5) *High inward-edge ratio seeding*: the inward-edge ratio for a vertex v is defined by the fraction of links connecting to another vertex inside the target community C among all the links coming out from v. We pick $|\mathcal{S}|$ vertices with inward-edge ratio ranked in the top one third among all vertices in C.

We show the effect of various seeding methods on LFR dataset ($mu = 0.1$) in Figure 6 and real datasets in Figure 7, respectively. It is interesting to note that the high-degree seeding method consistently achieves the higher F1 score than low-degree seeding, random seeding, and triangle seeding methods. Triangle seeding leads to the worst performance with low F1 score and high standard deviation. This implies that seeding from a compact core structure is less advantageous than seeding sporadically among vertices. The intuitive explanation behind this phenomenon is that it is more difficult for the probabilities to spread out when the random walk initiates from a cohesive structure.

Another interesting observation is that high inward-edge ratio seeding method can consistently lead to the best performance among different seeding methods on both synthetic and real datasets. To explain this, when a seed mostly links to vertices within the same community, random walks starting from this type of seeds would more likely transit probabilities into vertices within the community rather than spreading out to vertices outside the community. A higher detection accuracy can be thus achieved since the target community contains much of the probability after short random walks.

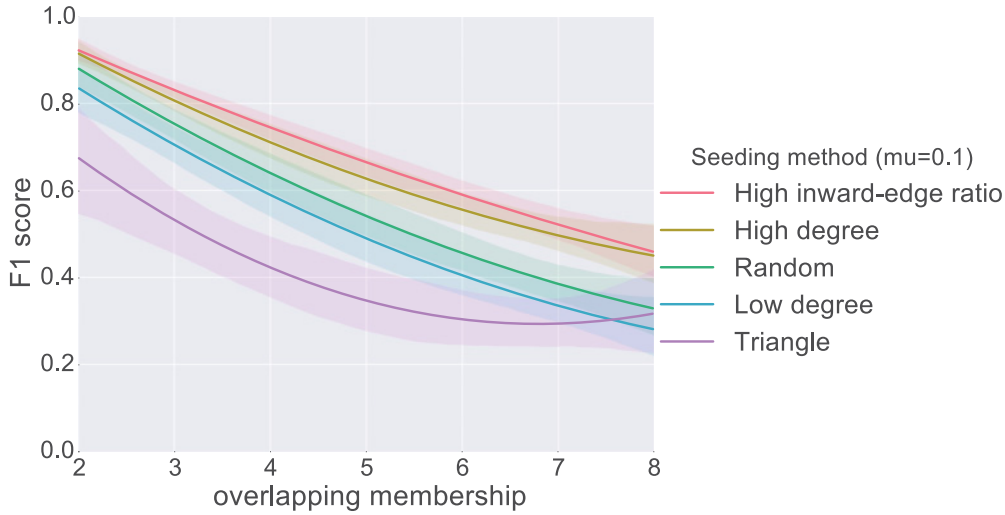

Fig. 6. The average F1 score on LFR datasets ($mu = 0.1$) with different seeding methods.

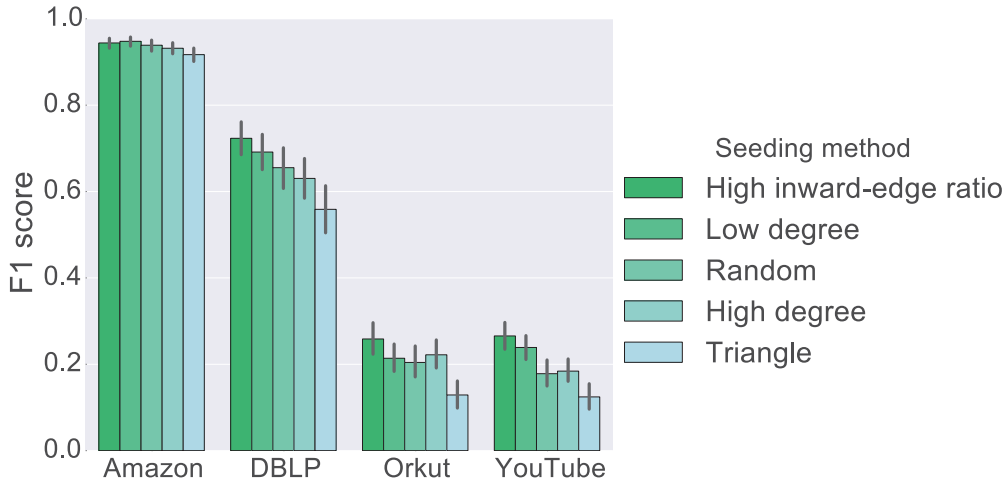

Fig. 7. The average F1 score on real datasets with different seeding methods.

Moreover, it is also striking to note the difference between the test results on synthetic datasets and that on real datasets. Even though the high-degree seeding method can always bring higher performance than that of random seeding on synthetic datasets, the behavior of these seeding methods on real networks is quite different. In Figure 7, we see that low-degree seeds lead to better result than that of high-degree seeds on DBLP and YouTube datasets. The degree of seeds does not have a significant impact on the performance in Amazon and Orkut networks since the performance of high-degree seeding and low-degree seeding almost tie with each other on these datasets. In [14], the authors compared the detection accuracy of PageRank-based SSE algorithm with high-degree seeding and random seeding on real networks, and concluded that random seeding method always outperforms high-degree seeding in all domains of real networks. However, we remark here that this observation does not apply to our algorithm as we find that high-degree seeding works slightly better than random seeding on Orkut and YouTube datasets.

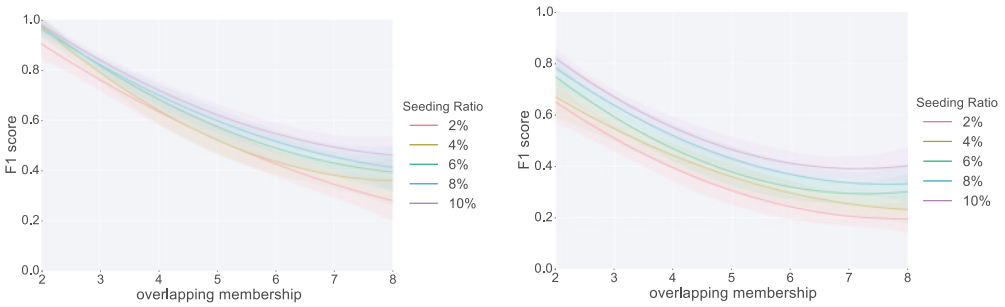

Fig. 8. The average F1 score on LFR benchmark data with different seeding ratio. The left figure corresponds to the datasets with mixing parameter $mu = 0.1$ and the right one corresponds to $mu = 0.3$.

For all the remaining analysis, we will be using *random seeding* due to its generality and practicality.

5.2 Seed Set Size

It is also interesting to investigate how the size of the seed set affects the performance of our algorithm.

We first experiment on the LFR benchmark datasets with varying seed set size. We choose seed set of size proportional to the size of the target community C. Specifically, we test with five different seeding ratios r: 2%, 4%, 6%, 8%, and 10%, respectively, and round $r \cdot |C|$ to an integer if it is a fraction. Figure 8 shows the F1 scores when $mu = 0.1$. The algorithm's performance can be improved in general as the seed set size increases. In the case when both mixing parameter and overlapping membership are small, e.g., $mu = 0.1, om = 2$, increasing the seed set size does not seem to affect the performance significantly, and seed set consisting of a small percentage of vertices are sufficient to discover the target community with high accuracy. This implies that when the structure of a small community is well-defined, our algorithm only needs 2 to 3 seeds to reveal the remaining members in a community of size roughly 100. In general, we use an 8% fraction of the vertices in the target community for the whole LFR datasets.

We conduct similar experiment on the real datasets. The result on real networks is interesting because increasing the seed set size has little effect on the performance. In particular, using only three seeds can yield almost the same performance as using an 8% fraction of the vertices in the target community as seeds on real datasets.

Our algorithm is therefore advantageous to many other SSE algorithms that require a higher fraction of vertices to be known. For example, in [14], the authors perform a similar experiment on DBLP network. The performance of their algorithm achieves the maximum recall of 0.3 when seeding ratio is 10%, while LEMON can achieve an average F1 score of 0.66 with 3 vertices. This makes our algorithm practical for real networks when it is impossible to collect a large number of seeds.

5.3 Further Extension

As the results of using different seeding methods suggests, high-degree seeds can heuristically lead to better result on synthetic data. Such heuristic implies that a vertex with higher degree may exert higher impact on shaping the subspace we are looking for, and thus affect the performance by leading to different sparse vectors where we obtain the "candidates" of the target community from.

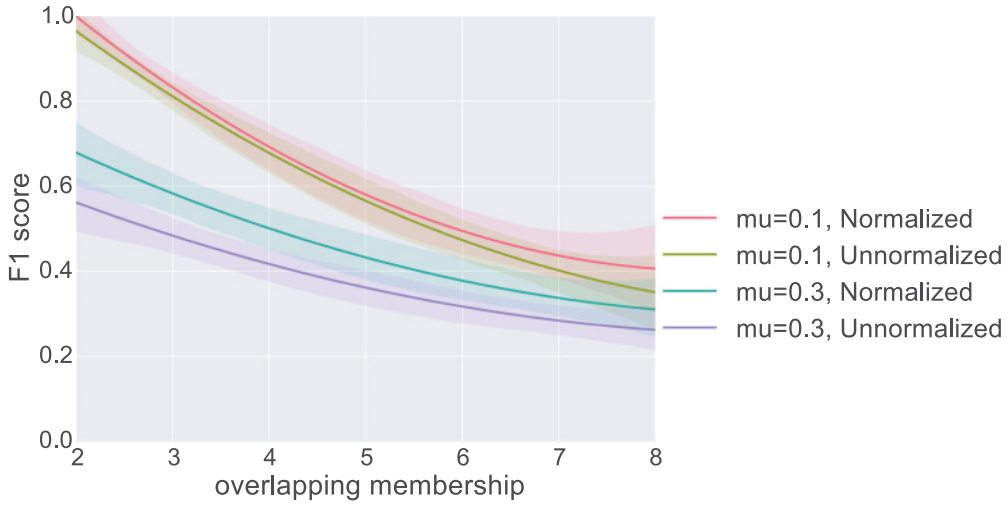

Fig. 9. Comparison of the average F1 score on LFR datasets with and without normalizing the initial probability vector by each seed's degree.

In practice, we usually have little control on the seed set. The chance we get a seed set of high-degree members is rare. More often than not, the degree of seeds is randomly distributed. We are therefore inspired to tailor our algorithm accordingly in order to emphasize the seeds with high degree. The modification is rather straightforward: when calculating the initial probability vector p_0 to start a random walk from, instead of evenly distributing the amount of probability to each seed, we initialize the probability vector according to the degree of each seed. Formally,

$$p_0(v_i) = \begin{cases} d(v_i)/\mathrm{Vol}(\mathcal{S}) & \text{if } v_i \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where $d(v_i)$ denotes the degree of vertex v_i. In other words, we enforce a bias towards the high-degree vertices at the beginning of the random walk. Note that each time after the reseeding process, the initial probability vector also needs to be recalculated in the same way.

Figure 9 depicts the experimental results on LFR benchmark graphs with and without degree normalized initialization for the random walk, respectively. We can find that degree-normalization of the initial probability vector results in better performance.

We then perform the same experiments on real networks, and find that degree-normalization would on the contrary, lead to slightly worse performance (see Figure 10). The completely different behavior of using degree normalization on real datasets is rather intriguing. To explain this intuitively, we can refer to the observation in Section 5.1 that on real datasets, a high-degree seed set is less advantageous than random seeds. By putting heavier weight on those high-degree vertices, the degree-normalization would unnecessarily worsen the performance on real datasets. We provide more discussions in explaining the effect of high-degree nodes in Section 6.4.

5.4 Enlarging the Initial Seed Set

In Section 5.2, we see that a larger seed set would lead to better results in general on synthetic datasets. But in the situation when there are not many seeds available, can we still find a way to improve the performance on synthetic datasets? This can be achieved via preprocessing the seed set before running our algorithm. Specifically, for each pair of vertices (v_i, v_j) in the seed

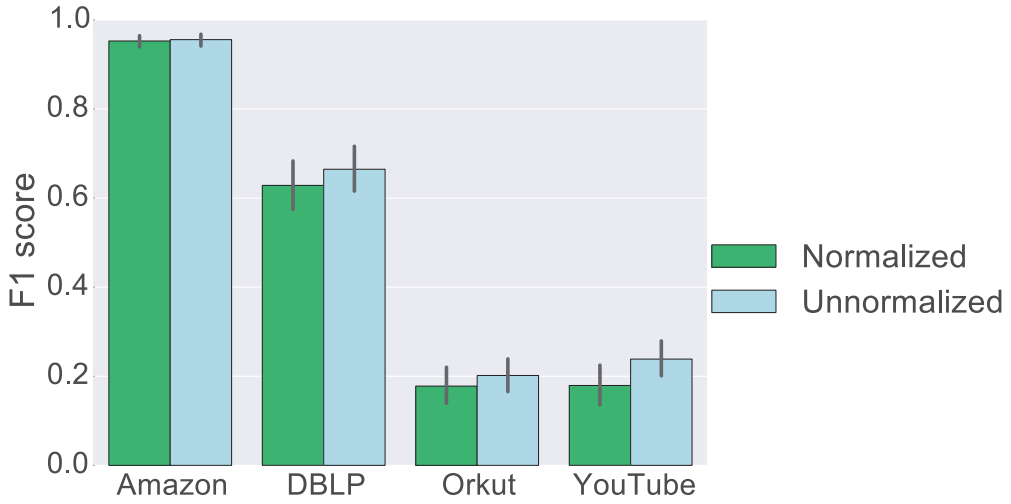

Fig. 10. Comparison of the average F1 score on real datasets with and without normalizing the initial probability vector by each seed's degree.

set $\mathcal{S}$, we search for the shortest path $\mathcal{P}$ connecting v_i and v_j, and add the vertices on the path to the original seed set if the length of the shortest path $|\mathcal{P}| \leq 3$. The intuition behind this idea is that any two seeds in the same community must be related for some reason, and they connect with each other either via a direct link or via some other intermediate vertices. In the latter case, those intermediate vertices bridging the seeds are also likely to be in the target community because they serve as the relational "relay" in order for the seeds to be in the same community.

Note that the procedure of enlarging the initial seed set $\mathcal{S}$ differentiates from the reseeding process while running the algorithm. The can be viewed as a pre-processing step before we feed the seed into the algorithm, which is used for the purpose of increasing the size of initial seed set. The running time used for enlarging the initial seed set is almost negligible.

In Figure 11, we compare the performance on LFR benchmark data with and without enlarging the initial seed set (with three randomly selected nodes from $|C|$). We can see that enlarging the seed set can statistically improve the performance. This method can help solve the dilemma of lacking enough available seeds, e.g., when the seed set consists of only 3 or 4 vertices.

6 COMPARISON WITH THE STATE-OF-THE-ART ALGORITHMS

In this section, we compare LEMON with several state-of-the-art approaches. For generality, we use plain LEMON algorithm without degree normalization or seed set enlargement. For real datasets, we use random seeding method with three initial seeds. For LFR benchmark graphs, we randomly sample 8% vertices from the target community as seeds.

6.1 Local Spectra vs. PageRank

The local spectra clustering approach and PageRank algorithm both utilize short random walks to detect the local community structure. PageRank is solely based on the single probability vector, and the latent community members are selected through ranking the probability value among vertices. The local spectral clustering advances PageRank-like algorithms by forming a subspace

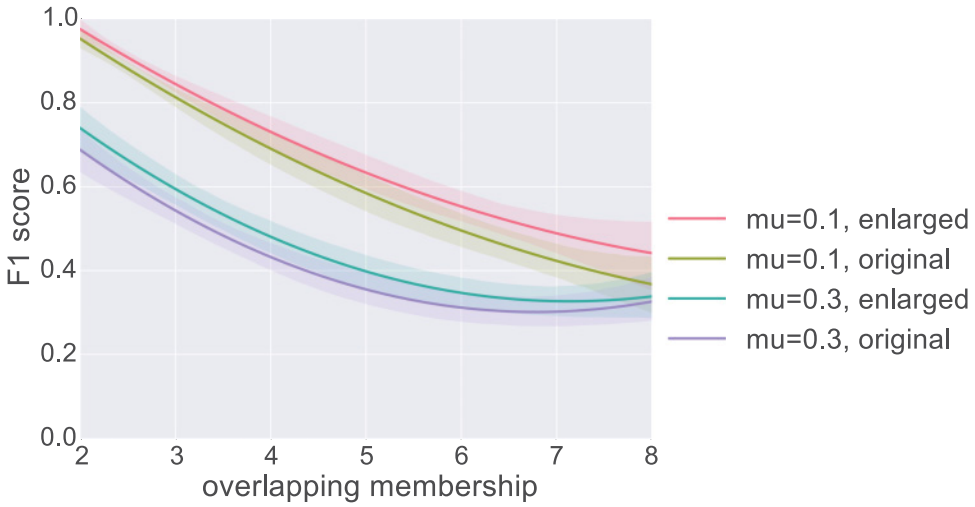

Fig. 11. Comparison of the average F1 score with and without enlarging the initial seed set on LFR benchmark datasets.

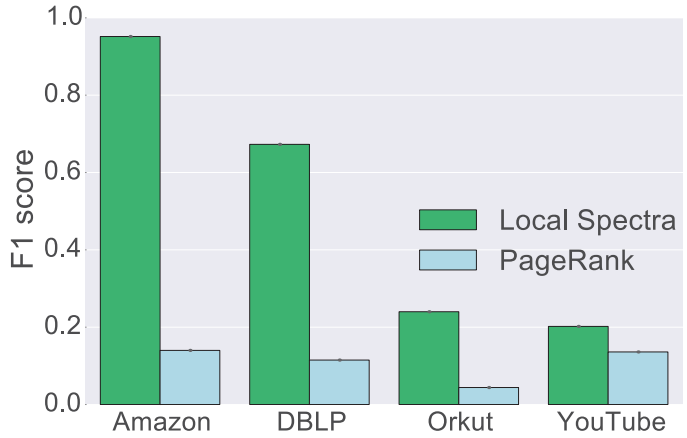

Fig. 12. Comparison of the average F1 score with local spectral clustering and PageRank algorithms. Height of the bars shows the mean over different communities.

based on the short random walk, and seeking for a sparse vector such that the seeds are in its support.

By comparing the performance of these two approaches on the real datasets, we show that seeking for the sparse vector is more effective than directly sorting the probability vector alone. Figure 12 shows the comparison of average F1 score obtained by local spectral clustering and PageRank, respectively.[10] From the result, we see that the performance gain brought by the local spectral method is significant, where it achieves more than five times higher accuracy on Amazon, DBLP, and Orkut networks. We also take into account of a variety of state-of-the-art community detection algorithms for performance comparison in Section 6.

[10]The statistical results of PageRank algorithm is sourced from [12].

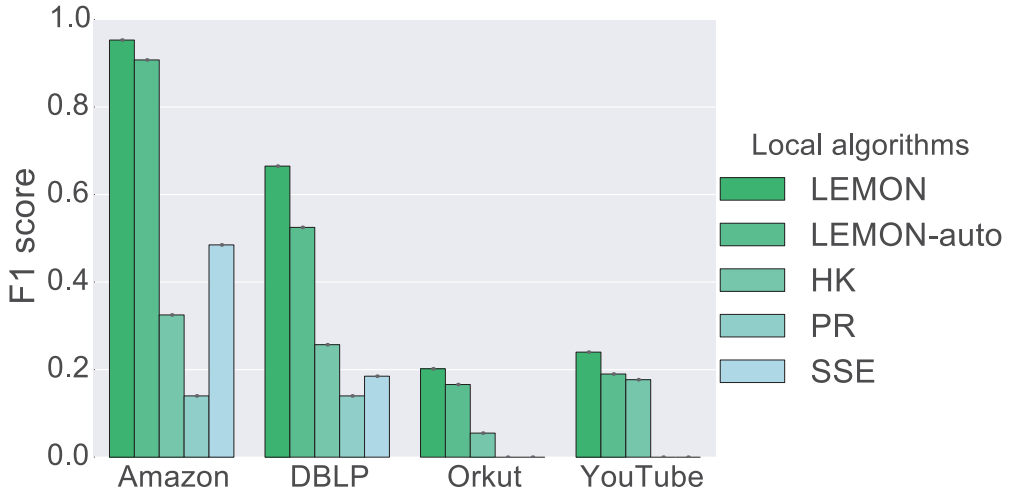

Fig. 13. Comparison of the average F1 score with state-of-the-art local detection algorithms on real networks.

To give a well-rounded performance comparison with state-of-the-art algorithms, we further compared our results to three localized community detection algorithms and four global community detection algorithms.

6.2 Comparison with Localized Algorithms

We compare with three locally based methods, Heat Kernel (HK) [12], PageRank (PR) [14], and SSE [29].

(1) Heat Kernel [12]: The heat kernel[11] (HK) is a type of graph diffusion for locally identifying a community nearby a starting seed node. The algorithm can deterministically find the community by computing the diffusion.
(2) PageRank [14]: The personalized PageRank (PR) scheme is computed using the power method and jumpback probability $\alpha = 0.10$ in [14].
(3) Seed Set Expansion [29]: The SSE method starts with searching for a good seed set, and then use personalized PageRank to expand the seeds until a community with optimal conductance is found.

Figure 13 illustrates the comparison of F1 scores on Amazon, DBLP, YouTube, and Orkut datasets. We use "Lemon-auto" to denote the results obtained by applying the stop criteria in Section 4.5. For each of the baseline method, we refer to the original results reported in this article. Since the results on Orkut and YouTube datasets are missing in [29] and [14], we use empty bars to indicate them.

Figure 13 shows that Lemon achieves an F1 score of 0.910 on the Amazon dataset, far outperforming the other algorithms. The average F1 scores increases the performance by three times compared with the heat kernel algorithm [12] on Amazon, DBLP, and Orkut networks. To compare with [29], we find that the average F1 score of our algorithm doubles their best performance achieved by the "spread hubs" method on Amazon dataset and triples the performance on the DBLP network. The performance comparison of Lemon and PageRank has been elaborated in

[11] https://www.cs.purdue.edu/homes/dgleich/codes/hkgrow.

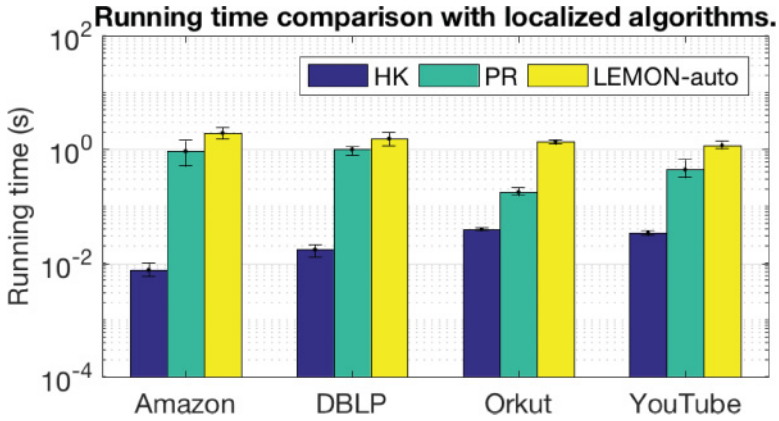

Fig. 14. Comparison of the running time with local random walk based detection algorithms on real networks.

Section 6.1. Also, note that in [14], the authors did not have an explicit stop criterion and instead having a pre-determined budget for the size of the target community. We compare with the F1 score at a budget of 100 for both Amazon and DBLP datasets. From the results on Amazon networks in [14], we notice that even granted a budget of 400, which is far beyond the average community size of 39 in Amazon network, only a recall of 0.45 can be achieved. And, we infer the F1 score would be even lower than this value since the precision is dragged down by the large budget set.

It is also worth noting that we only use three randomly picked seeds for all the test cases on each dataset. Our algorithm requires very fewer seeds than other algorithms such as [14].

We further compare the running time with localized random walk based algorithms, namely Heat Kernel [12] and Personalized PageRank [14]. Figure 14 shows the running time of different approaches (in log scale) on four real datasets in consideration.[12] We notice LEMON-auto takes slightly longer time compared to other two walk-based approaches, but not substantial. This is due to the component of solving linear programming takes extra computation time.

The experiment has verified that our algorithm is able to achieve high accuracy on large networks constituting communities of average size roughly hundred. This implies that our approach is well-suited for the task of detecting small communities in large networks.

6.3 Comparison with Global Algorithms

We also compare local spectral clustering with several state-of-the-art global based algorithms.

(1) OSLOM [16]:
OSLOM[13] is based on the optimization of a fitness function expressing the statistical significance of clusters with respect to random fluctuations (i.e., the random graph generated by the configuration model [22] during community expansion). The worst case running time of OSLOM is $O(n^2)$.

(2) DEMON [8]:
The DEMON[14] algorithm adopts a local-first approach for finding communities. It democratically lets each vertex vote for the communities it sees surrounding it in its limited

[12]Results are referenced from a followup work [13] done by the first and third author. Algorithms are implemented in MATLAB.
[13]http://www.oslom.org/software.htm.
[14]http://www.michelecoscia.com/?page_id=42.

Table 5. Comparison of Accuracy with Global Algorithms on Real Datasets

Algorithm	Implementation	Amazon	DBLP	YouTube	Orkut
LEMON	Python	0.953	0.665	0.240	0.202
LEMON-auto	Python	0.910	0.525	0.190	0.170
DEMON	Python/C++	0.164	0.196	0.031	–
OSLOM	C++	0.766	0.542	–	–
LC	Python/C++	0.815	0.527	–	–

Table 6. Comparison of Running Time with Global Algorithms

Algorithm	Implementation	Amazon	DBLP	YouTube	Orkut
LEMON	Python	<15s	<15s	<15s	<15s
LEMON-auto	Python	<15s	<15s	<15s	<15s
DEMON	Python/C++	4,562s	727,675s	22,395s	–
OSLOM	C++	885,867s	23,262s	>10d	–
LC	Python/C++	4,606s	49,045s	>10d	–

view of the global system using a label propagation algorithm, and then merges the local communities into a global collection.

(3) LC [2]:
LC[15] is a global partitioning algorithm that first builds a hierarchical link dendrogram according to the link similarity and then cuts the dendrogram at some threshold to yield link communities. The time complexity is $O(nk_{\max}^2)$, where $k_{\max}$ is the maximum vertex degree in the network.

Tables 5 and 6 summarize the average F1 score as well as the running time of each algorithm on real datasets. Among the baselines, OSLOM and LC fail to terminate within 10 days on the YouTube dataset. The OSLOM algorithm can achieve rather good performance but does not scale well. In contrast, our algorithm can consistently return the result within few seconds irrespective of how large the entire graph is. Besides, our algorithm has small memory consumption, and a machine with 4GB RAM can afford to process networks as large as Orkut since the algorithm does not have to store the whole graph in memory. Moreover, our locally based algorithm is parallelizable because each SSE can be computed independently. Such property can bring a further performance gain on running time with multi-threaded implementation [29].

In Figures 15 and 16, we compare the average F1 score with state-of-the-art algorithms on LFR benchmark graphs. During the experimentation, we also incorporate the methods that can effectively improve the performance on synthetic datasets that are addressed in Section 5.3. We notice that our algorithm outperforms the baseline algorithms even when we use the random seeding strategy. When the mixing parameter $mu = 0.3$, as is shown in Figures 15 and 16, LEMON brings about 30%–40% relative improvement compared with the best results among the baselines. And we can expect the performance gain to be even more significant if the seeds possess the qualities discussed in Section 5.1.

Among the four baselines, we notice that LC and DEMON consistently perform poorly on both groups of the synthetic datasets. We further look into the communities found by LC and DEMON, respectively, and find that LC tends to partition the graphs into very small pieces while DEMON, on the contrary, usually finds communities that are much larger than the ground truth communities.

[15] https://github.com/bagrow/linkcomm.

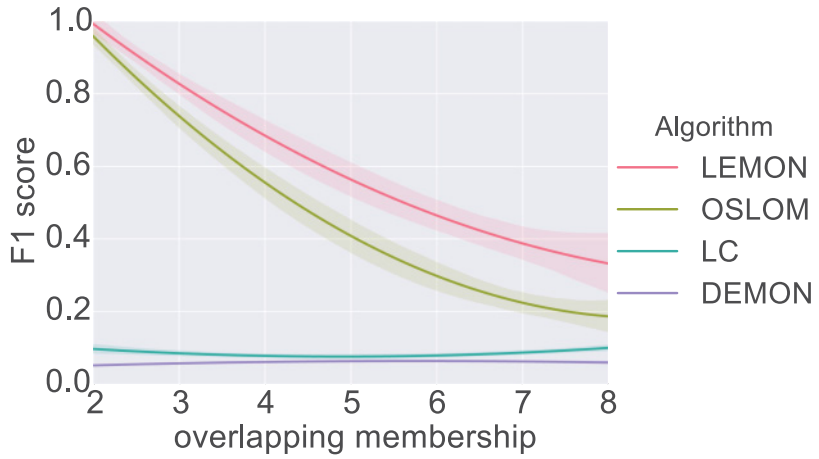

Fig. 15. Comparison of the average F1 score on LFR datasets ($mu = 0.1$) with baseline algorithms.

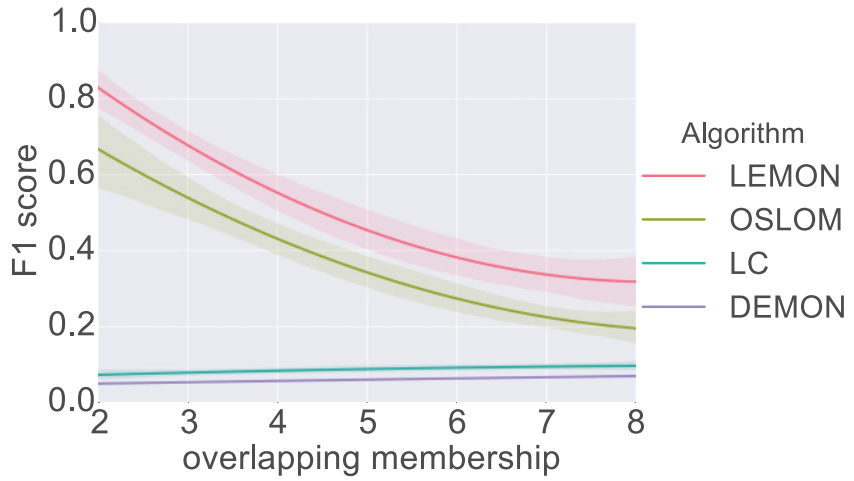

Fig. 16. Comparison of the average F1 score on LFR datasets ($mu = 0.3$) with baseline algorithms.

This implies that both algorithms extract structures from networks that bear little resemblance to the natural formation of the communities. However, we remark here that even LC fails to recognize the communities well on the synthetic data, it perform better on real datasets as we see in Table 6.

6.4 Empirical Comparison Between Synthetic and Real Data

Networks are not all similar, and we cannot assume one algorithm works for finding communities in a network will behave the same on the other networks. Therefore, it is important to develop the understanding of how different types of networks affect the behavior of algorithms.

Our algorithm sustains a consistent performance on both LFR benchmark graphs and real networks though, we still want to summarize and call the attention to several subtle differences here.

First, LEMON is less sensitive to the parameter of random walk step k and subspace dimension l on real networks than that on LFR benchmark graphs. In practice, fixing (k, l) to be $(3, 3)$ for real networks can ensure a good performance.

Second, LEMON is less sensitive to the seed set size on real networks than that on LFR benchmark. In practice, a seed set size of 3 (independent of actual ground truth community size) can guarantee a good performance on real networks. As for LFR, we adopt the seed set size to be proportional to the community size (8%).

Third, LEMON is more sensitive to the high-degree seeds in graphs where the node distribution is highly skewed (e.g., when the node degree can range from tens to thousands). In LFR graphs, the degree of a vertex is at most 50. Whereas in some large real networks such as YouTube, the degree of a vertex can range from 1 to >1,000, making the degree distribution much more skewed than that seen in LFR graphs. And, we expect that vertices with unusually high degree in real networks would have a stronger power in controlling the trend for the probabilities to spread out during the random walk, and thus have a higher risk to enter some other neighboring communities. Such an effect can be counterbalanced by putting less initial probabilities on these "super core."

The above empirical analysis informs us that finding communities in real networks seems to be less parameterized than that on synthetic datasets for our algorithm. This indicates that our algorithm is better suited for uncovering those naturally well-formed communities than the artificially constructed communities in practice.

7 CONCLUSION

The problem of identifying small community structure in large networks has been gaining importance. In this article, we have presented a method for finding overlapping communities by seeking a sparse vector in the span of local spectra where the seeds are in its support. To overcome the drawbacks of traditional spectral clustering methods, we propose a novel method to construct the local spectra based on the singular vector approximations drawn from short random walks. Our algorithm enables finding a small community in time functional to the size of the community, and it consistently returns the result within seconds even for a network with millions of vertices. We demonstrate the effectiveness and efficiency of our method for discovering communities on both synthetic and real-world datasets. As the experimental result shows, our algorithm achieves the highest detection accuracy among the state-of-the-art proposals.

Many other fundamentally important research questions remain to be addressed. First, the community detection algorithm based on local spectral clustering could be potentially applied to the membership detection problem, i.e., finding all the communities that an arbitrary vertex belongs to. Second, during SSE, we adopt the first low-conductance community as the target community, which usually yields a high resemblance to the ground truth community. It would also be interesting to look further into some larger low-conductance communities and see if a hierarchical structure exists. In this case, some large social groups consisting of several small communities are likely to be discovered.

REFERENCES

[1] Bruno Abrahao, Sucheta Soundarajan, John Hopcroft, and Robert Kleinberg. 2014. A separability framework for analyzing community structure. *ACM Transactions on Knowledge Discovery from Data* 8, 1 (2014), 5.

[2] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *Nature* 466, 7307 (2010), 761–764.

[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *Proceedings of FOCS*. IEEE, 475–486.

[4] Reid Andersen and Kevin J. Lang. 2006. Communities from seed sets. In *Proceedings of WWW*. ACM, 223–232.

[5] Andrea L. Bertozzi and Arjuna Flenner. 2012. Diffuse interface models on graphs for classification of high dimensional data. *Multiscale Modeling & Simulation* 10, 3 (2012), 1090–1118.

[6] Xavier Bresson, Huiyi Hu, Thomas Laurent, Arthur Szlam, and James von Brecht. 2014. An incremental reseeding strategy for clustering. *arXiv preprint arXiv:1406.3837* (2014).
[7] Fan R. K. Chung. 1997. *Spectral Graph Theory*. Vol. 92. American Mathematical Soc.
[8] Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. 2012. Demon: A local-first discovery method for overlapping communities. In *Proceedings of KDD*. ACM, 615–623.
[9] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3 (2010), 75–174.
[10] Kun He, Yiwei Sun, David Bindel, John Hopcroft, and Yixuan Li. 2015. Detecting overlapping communities from local spectral subspaces. In *Proceedings of ICDM*. ACM.
[11] Di Jin, Bo Yang, Carlos Baquero, Dayou Liu, Dongxiao He, and Jie Liu. 2011. A Markov random walk under constraint for discovering overlapping communities in complex networks. *Journal of Statistical Mechanics: Theory and Experiment* 2011, 5 (2011), P05031.
[12] Kyle Kloster and David F. Gleich. 2014. Heat kernel based community detection. In *Proceedings of KDD*. ACM.
[13] Kyle Kloster and Yixuan Li. 2016. Scalable and robust local community detection via adaptive subgraph extraction and diffusions. *arXiv preprint arXiv:1611.05152* (2016).
[14] Isabel M. Kloumann and Jon M. Kleinberg. 2014. Community membership identification from small seed sets. In *Proceedings of KDD*. ACM.
[15] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78, 4 (2008), 046110.
[16] Andrea Lancichinetti, Filippo Radicchi, José J. Ramasco, and Santo Fortunato. 2011. Finding statistically significant communities in networks. *PloS One* 6, 4 (2011), e18961.
[17] Conrad Lee, Fergal Reid, Aaron McDaid, and Neil Hurley. 2011. Seeding for pervasively overlapping communities. *Physical Review E* 83, 6 (2011).
[18] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Statistical properties of community structure in large social and information networks. In *Proceedings of WWW*. ACM, 695–704.
[19] Yixuan Li, Kun He, David Bindel, and John E. Hopcroft. 2015. Uncovering the small community structure in large networks: A local spectral approach. In *Proceedings of WWW*. ACM, 658–668.
[20] Michael W. Mahoney, Lorenzo Orecchia, and Nisheeth K. Vishnoi. 2012. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *The Journal of Machine Learning Research* 13, 1 (2012), 2339–2365.
[21] Marina Meila and Jianbo Shi. 2001. Learning segmentation by random walks. *Advances in Neural Information Processing Systems*. 873–879.
[22] Michael Molloy and Bruce Reed. 1995. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms* 6, 2–3 (1995), 161–180.
[23] Andrew Y. Ng, Michael I. Jordan, Yair Weiss, and others. 2002. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems* 2 (2002), 849–856.
[24] Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. In *Proceedings of the International Symposium on Computer and Information Sciences (ISCIS'05)*. Springer, 284–293.
[25] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (2007), 036106.
[26] Martin Rosvall and Carl T. Bergstrom. 2011. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS One* 6, 4 (2011), e18209.
[27] Yousef Saad. 2003. *Iterative Methods for Sparse Linear Systems*. SIAM.
[28] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of STOC*. ACM, 81–90.
[29] Joyce Jiyoung Whang, David F. Gleich, and Inderjit S. Dhillon. 2013. Overlapping community detection using seed set expansion. In *Proceedings of CIKM*. ACM, 2099–2108.
[30] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys* 45, 4 (2013), 43.
[31] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of ICDM*. 10–13.
[32] Vinko Zlatić, Andrea Gabrielli, and Guido Caldarelli. 2010. Topologically biased random walk and community finding in networks. *Physical Review E* 82, 6 (2010), 066109.

Received September 2015; revised February 2017; accepted June 2017