On Multi-Query Local Community Detection

Yuchen Bian [#1], Yaowei Yan [#2], Wei Cheng [*3], Wei Wang [†4], Dongsheng Luo [#5], Xiang Zhang [#6]
[#] College of Information Sciences and Technology, The Pennsylvania State University; [*] NEC Laboratories America;
[†] Department of Computer Science, University of California, Los Angeles
[1,2,5,6]{yub31,yxy230,dul262,xzhang}@ist.psu.edu; [3]weicheng@nec-labs.com; [4]weiwang@cs.ucla.edu

***Abstract*—Local community detection, which aims to find a target community containing a set of query nodes, has recently drawn intense research interest. The existing local community detection methods usually assume all query nodes are from the same community and only find a single target community. This is a strict requirement and does not allow much flexibility. In many real-world applications, however, we may not have any prior knowledge about the community memberships of the query nodes, and different query nodes may be from different communities. To address this limitation of the existing methods, we propose a novel memory-based random walk method, MRW, that can simultaneously identify multiple target local communities to which the query nodes belong. In MRW, each query node is associated with a random walker. Different from commonly used memoryless random walk models, MRW records the entire visiting history of each walker. The visiting histories of walkers can help unravel whether they are from the same community or not. Intuitively, walkers with similar visiting histories are more likely to be in the same community. Moreover, MRW allows walkers with similar visiting histories to reinforce each other so that they can better capture the community structure instead of being biased to the query nodes. We provide rigorous theoretical foundation for the proposed method and develop efficient algorithms to identify multiple target local communities simultaneously. Comprehensive experimental evaluations on a variety of real-world datasets demonstrate the effectiveness and efficiency of the proposed method.**

I. Introduction

Local community detection (or local clustering) [2]–[8] has recently attracted much attention due to its fundamental importance in large network analysis. Given a network and a set of query nodes, the goal is to find a community (or cluster) that the query nodes belong to. Random walk based methods have been shown to be effective in capturing local community structures and routinely used in this task [3]–[5].

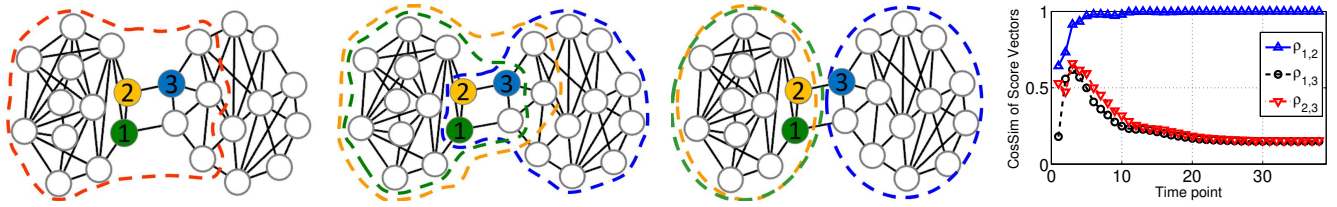
Despite of their success, most existing local community detection methods assume that all query nodes are from the same target community, and thus only aim to identify a single community to include all query nodes [2], [5], [6], [9], [10]. This is a stringent requirement and does not allow much flexibility. However, in many real-world applications, the query nodes may be in different communities and we often have no prior knowledge about community memberships of the query nodes. For example, in genetics, many complex diseases are caused by multiple genetic factors that belong to different biological pathways (which can be modelled as communities in biological networks) [11]–[13]. Thus given several disease candidate genes as the query nodes, we cannot simply assume that all of them belong to the same pathway and identify a single pathway to include all candidate genes. As another example, in microscopic image retrieval [14], the query images may belong to different disease subtypes and we need to identify multiple image clusters (communities) that correspond to different disease subtypes in the image similarity network. Similarly, in a collaboration network where nodes represent researchers, a number of interested researchers may belong to different research communities even if they collaborate on the same research project [15]. Therefore, an ideal local community detection method should provide the flexibility to allow certain query nodes to belong to different communities.

Figure 1(a) shows an example network with three query nodes from two different communities. Nodes ① and ② are from the left community and node ③ is from the right community. A straightforward solution is to assume that all three query nodes are in the same community and to apply existing local clustering methods to find the community that they belong to. In Figure 1(a), the red dotted curve shows the local community identified by the classic PageRank-Nibble algorithm [2]. PageRank-Nibble adopts a lazy random walk approach which gradually expands from the query nodes and finds the cluster with the minimum conductance as the target community. As we can see, the identified community includes nodes from both clusters. Thus, we cannot simply assume all query nodes are from the same community in this situation.

Since we have no prior knowledge of community memberships of the query nodes, an alternative solution is to separately detect the target community for each query node. We refer to this approach as the *basic approach*. Figure 1(b) shows the result of applying PageRank-Nibble to detect communities for the three query nodes separately. The detected communities are represented by dotted curves with colors corresponding to the colors of the query nodes. We can observe that the detected communities are biased toward their corresponding query nodes. Some neighbors of the query node that are outside of the target community are also included in the result.

There are two key limitations in the basic approach described above. The first is that the random walkers tend to be trapped in the neighbourhood of the query nodes. This is common in the existing methods [2], [5], [6], since the walkers do not have any memory of their past visiting histories and they often need to jump back to the query nodes and restart [16]. Another limitation is that each walker finds its own community independently and there is no interaction between the walkers.

978-1-5386-9159-5/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDM.2018.00016



(a) Result of PageRank-Nibble (assuming all query nodes are in the same community)

(b) Result of PageRank-Nibble (detecting community for each query node separately)

(c) Result of MRW (detecting communities simultaneously for all query nodes)

(d) Cosine similarity of the visiting history score vectors of the three walkers in MRW

Fig. 1. An example network with three query nodes from two communities

To address the limitations of the existing local community detection methods, in this paper, we propose MRW, a *memory-based random walk* model, to simultaneously identify multiple target local communities for a given set of query nodes. MRW does not assume all query nodes are from the same community. Similar to the basic approach described above, we use one walker for each query node. The difference is that, in MRW, we (1) record the visiting history for each walker, and (2) introduce interactions among the walkers based on their visiting histories. In MRW, each walker is associated with a score vector representing the visiting history of that walker. More specifically, we use a sliding window to record the key positions of previous steps and aggregate the entire visiting history for each walker. The visiting history score vectors can then be utilized to introduce interactions between the walkers. In particular, walkers with similar visiting histories can mutually reinforce each other so that their score vectors can better represent the community structures instead of being biased to the query nodes.

Figure 1(c) shows the detected local communities by MRW in the example network. We can see that MRW successfully identified the two target communities that the three query nodes belong to. To gain further insight about the intermediate steps of MRW, Figure 1(d) shows the cosine similarity of the visiting history score vectors of the three walkers. As we can see, the cosine similarity between every two score vectors corresponding to the queries in the same community, i.e., nodes ① and ②, becomes stronger during the process, while the cosine similarity between the score vectors of the queries in different communities, i.e., node-pair ① and ③, and node-pair ② and ③, becomes weaker. This shows the effect of mutual reinforcement between the walkers based on their visiting histories. It helps to alleviate the query bias issue in traditional local community detection methods.

The rest of this paper is organized as follows. Section II summarizes the related work. Section III-A introduces the memory-based random walk method MRW for a single query node. Section III-C discusses how multiple random walkers can mutually reinforce each other and improve the performance. Section IV presents algorithms to compute the score vectors and to find the target local communities. Section V shows the results of comprehensive experimental evaluations. Section VI gives the concluding remarks.

II. Related Work

Basic random walk models. Many existing local community detection methods are based on random walks. The PageRank-Nibble algorithm [2] uses a lazy random walker to explore a graph and ranks nodes by the degree-normalized visiting probabilities. It then sweeps the ranking list to find the subset of top-ranked nodes with the minimum conductance. The heat kernel method [6] replaces the decay factors in PageRank-Nibble with heat diffusion parameters to find the local cluster. In [9], comprehensive experimental evaluations suggest that random walk with restart is one of the most effective models to find the local community. The query-biased local clustering method [5] assigns weights to nodes based on the visiting probabilities of a random walker. The query-biased densest subgraph is then identified and considered as the target community. Random walk models are also used to detect motif patterns around the query node [17], [18]. However, all these methods use only one walker to explore the network and have no record the visiting history.

Advanced random walk models. Recently, several more advanced methods have been developed to address the limitation of basic random walk models. In [16], the authors show that traditional memoryless random walk models are usually biased to the query nodes and propose a second-order random walk method. However, this method only memorizes one immediate step before current step and does not leverage the history before that. A multi-agent random walk method [7] is developed to find local community by restricting the distance between different agents. In [19], [20], the authors developed a double-walker model (the background walker and the foreground walker) for bilayer image segmentation. The multi-walker chain method [8] treats multiple walkers as an integrated group to find a single target local community. Nevertheless, the walkers in this method are also memoryless.

Other methods for local community detection. In addition to random walks, spectral clustering has also be used for local community detection [21]–[23]. The general idea is to project the spectral space to a low dimensional space and identify the local community in the projected subspace. The tensor spectral clustering method [24] takes advantages of higher-order structures such as triangles to improve the accuracy of the detected local community. The localized Label propagation method [25] propagates the label from a query node until convergence to form a community. Optimization based algorithms have also

TABLE I
SYMBOLS AND DEFINITIONS

Symbol	Definition
$G = (V, E)$	graph G with node set V and edge set E
$\mathbf{P}$	transition matrix
Q, q_i, S	query node set Q; query node $q_i \in Q$; $S = \|Q\|$
α, β, γ	α, β are decay factors; γ is the reinforcement factor
K	sliding window length
$\mathbf{x}_i^{(t)}$	score vector of q_i at time point t
$\mathbb{x}_i^{(t)}$	reinforced score vector of q_i at time point t
$\mathbf{v}^{(t)}$	visiting history vector at time point t
$\mathbf{e}^{(t)}$	vector for key positions at time point t
$\hat{\mathbf{R}}$, $\mathbf{R}$	$\hat{\mathbf{R}}$: reinforcement matrix; $\mathbf{R}$: column-normalized $\hat{\mathbf{R}}$

been developed for local community finding. In these methods, the local community structure is usually captured by certain measures such as local modularity [26], k-clique [27]–[29], k-truss [30]–[32], and k-core [33], [34]. A comprehensive survey of these optimization based algorithms can be found in [35].

Note that most of the methods discussed above assume that the query nodes are from the same community and try to find a single target local community. None of them addresses the problem of finding multiple local communities for multiple queries simultaneously, which is the focus of this paper.

Connection Subgraph Finding. The problem of connection subgraph finding is related to local community detection [36]–[39]. Different from local community detection, the goal of connection subgraph finding is to identify a *small* number of additional nodes to connect the query nodes. For example, the center-piece subgraph problem [36] tries to find a subgraph to connect all query nodes with a budget b of additional nodes. In [37], the connecting nodes are selected by following the Minimum Description Length principle. The minimum Wiener connector is introduced in [38] to connect the query nodes. Most methods for connection subgraph finding require the returned subgraph to be connected. The method in [39] allows disconnected subgraphs. In this method, network inefficiency is used as the cohesiveness measure to select connecting nodes.

III. THE MRW MODEL

In this section, we first introduce MRW for a single query, and then discuss how to generalize it for multiple queries by introducing reinforcement between different walkers. Important symbols are listed in Table I.

A. Single Query MRW

Given an undirected and connected graph $G = (V, E)$, we use $\mathbf{W}$ to represent its adjacent matrix, where entry $\mathbf{W}(i, j)$ is the weight of edge $(i, j) \in E$. Let $\mathbf{P}$ represent the transition matrix, where $\mathbf{P}(i, j)$ is the transition probability from node i to j, i.e., $\mathbf{P}(i, j) = \mathbf{W}(i, j)/\sum_{j \in V} \mathbf{W}(i, j)$.

We first briefly review the well-known random walk with restart (RWR) model which has been routinely used in the existing local community detection methods [2], [5], [9], [16].

In RWR, starting from a query node q, a single walker randomly explores the network. At each time point, the walker has probability α ($0 < \alpha < 1$) to follow the transition probabilities in $\mathbf{P}$, and probability $(1 - \alpha)$ to jump back to q. Formally, we have

$$\mathbf{x}^{(t+1)} = \alpha \mathbf{P}^\intercal \mathbf{x}^{(t)} + (1 - \alpha)\mathbf{q} \tag{1}$$

where $\mathbf{x}^{(t)}$ is the node visiting probability vector at time t, and $\mathbf{q}$ is a vector with value 1 for the q-th entry and 0 for all others. Note that RWR is memoryless since the next step of the walker only depends on the current node and the query node q.

The key difference between our method MRW and RWR is that in MRW we use a sliding window to memorize the key positions that the walker has previously visited and aggregate the entire visiting history of the walker. Moreover, the next step of the walker depends not only on the current node and q but also where the walker has visited. Specifically, in MRW, we have

$$\mathbf{x}^{(t+1)} = \alpha \mathbf{P}^\intercal \mathbf{x}^{(t)} + (1 - \alpha)\mathbf{v}^{(t)} \tag{2}$$

where $\mathbf{v}^{(t)}$ represents the aggregated history of the previous steps. Next, we discuss how to obtain $\mathbf{v}^{(t)}$.

At time t, we refer to the node(s) with the largest visiting probability as *key positions* of the walker. We use vector $\mathbf{e}^{(t)}$ to represent these key positions. More specifically, suppose that there are $n \geq 1$ key positions, the entries in $\mathbf{e}^{(t)}$ corresponding to the n key positions are set to $1/n$, and all other entries are 0.

To record the visiting history, we use a sliding window of length K to aggregate these key positions. That is,

$$\mathbf{v}^{(t)} = (1 - \beta^{t-1})\mathbf{v}^{(t-1)} + \beta^{t-1}\frac{1}{K}\sum\nolimits_{k=1}^{K} \mathbf{e}^{(t+1-k)} \tag{3}$$

where $\frac{1}{K}\sum_{k=1}^{K} \mathbf{e}^{(t+1-k)}$ represents the average of the key position vectors in the current time window, i.e., the past K steps. Intuitively, $\mathbf{v}^{(t)}$ combines the previous visiting history (represented by $\mathbf{v}^{(t-1)}$) and the average of key position vectors in the current time window with a tuning parameter β (0< β <1). Note that, initially, when $t < K$, $\mathbf{e}^{(t-K)}$ is set to be $\mathbf{e}^{(0)}$ and $\mathbf{v}^{(0)} = \mathbf{e}^{(0)}$, where $\mathbf{e}^{(0)}$ is the same as the vector $\mathbf{q}$ in RWR which represents the query node q.

B. Convergence

The following lemma shows that the difference between the score vectors at time points t and $(t + 1)$ will decrease when t becomes larger, thus the convergence of $\mathbf{x}^{(t)}$ is guaranteed.

Lemma 1:

$$\Delta^{(t+1)} \leq \begin{cases} \frac{2}{|\beta - \alpha|}\max\{\beta^t, \alpha^t\} & \text{if } \alpha \neq \beta \\ 2t\beta^{t-1} & \text{if } \alpha = \beta \end{cases}$$

where $\Delta^{(t+1)} = \| \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)} \|_1$.

Proof: Based on Equation (3), we have

$$\| \mathbf{v}^{(t)} - \mathbf{v}^{(t-1)} \|_1 = \beta^{t-1} || \mathbf{v}^{(t-1)} - \frac{1}{K}\sum_{k=1}^{K} \mathbf{e}^{(t+1-k)} ||_1 \leq 2\beta^{t-1}$$

Then according to Equation (2) and $\| \mathbf{P}^\intercal \|_1 = 1$, we have

$$\begin{aligned}
\Delta^{(t+1)} =& \| \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)} \|_1 \\
\leq& \alpha \| \mathbf{P}^\intercal(\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)}) \|_1 + (1-\alpha) \| \mathbf{v}^{(t)} - \mathbf{v}^{(t-1)} \|_1 \\
\leq& \alpha\Delta^{(t)} + (1-\alpha)2\beta^{t-1} \\
\leq& \alpha^t\Delta^{(1)} + 2(1-\alpha)\beta^{t-1}\sum_{m=0}^{t-1}(\frac{\alpha}{\beta})^m \\
\leq& 2[\alpha^t + (1-\alpha)\beta^{t-1}\sum_{m=0}^{t-1}(\frac{\alpha}{\beta})^m]
\end{aligned}$$

If $\alpha \neq \beta$, we have

$$\begin{aligned}
\Delta^{(t+1)} \leq& \frac{2}{\beta-\alpha}[(1-\alpha)\beta^t - (1-\beta)\alpha^t)] \\
\leq& \frac{2}{|\beta-\alpha|}\max\{\beta^t, \alpha^t\}
\end{aligned}$$

If $\alpha = \beta$, we have $\Delta^{(t+1)} \leq 2t\beta^{t-1}$. ■

The following theorem shows that, in practice, we can set a small tolerance value ϵ ($\epsilon > 0$) as the stopping criteria.

Theorem 1: There exists a constant t_c such that when $t > t_c$, $\Delta^{(t+1)} = \| \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)} \|_1 < \epsilon$ for a small tolerance $\epsilon > 0$.

Proof: From Lemma 1, if $\alpha \neq \beta$, $\Delta^{(t+1)} \leq \frac{2}{|\beta-\alpha|}\max\{\beta^t, \alpha^t\}$, then $t_c = \lfloor \frac{\log(\epsilon|\beta-\alpha|/2)}{\log(\max\{\alpha,\beta\})} \rfloor$; if $\alpha = \beta$, $\Delta^{(t+1)} \leq 2t\beta^{t-1}$, then $t_c = \lfloor \frac{W_{-1}(\epsilon\beta \ln \beta/2)}{\ln \beta} \rfloor$, where W_{-1} is the negative branch of the Lambert-W function [1]. ■

In practice, the score vector usually converges very fast. For example, when $\alpha = 0.2, \beta = 0.4, \epsilon = 10^{-3}$, t_c is about 10 on the real-world datasets used in our experiments.

C. Multi-Query MRW

Next, we discuss how to extend the basic MRW model to handle multiple queries simultaneously. Given a set of query nodes Q, for each query node $q_i \in Q$ ($1 \leq i \leq S, S = |Q|$), we assign a random walker to q_i. Let $\mathbf{x}_i^{(t)}$ be the visiting probability vector of the walker associated with q_i at time point t (as defined in Equation (2)). Intuitively, if two query nodes q_i and q_j are in the same community, their corresponding vectors $\mathbf{x}_i^{(t)}$ and $\mathbf{x}_j^{(t)}$ should have high similarity; otherwise their similarity should be low. Thus for walkers whose score vectors are highly similar, we allow them to reinforce each other.

Let $\hat{\mathbf{R}}^{(t)}$ be the similarity matrix of score vectors at time t. The initial value $\hat{\mathbf{R}}^{(0)} = \mathbf{0}$. When $t > 0$, for $i \neq j$, $\hat{\mathbf{R}}^{(t)}(i,j) = cos(\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)})$ if $cos(\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)}) > \theta$, where $cos(\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)})$ is the cosine similarity between the two vectors. Here we only consider the similarity values larger than a small threshold (e.g., $0 \leq \theta \leq 0.01$) since smaller similarity imply that there are no or only a small proportion of the entries in the two vectors with similar scores, which indicates that the two query nodes are in different communities.

[1]The Lambert-W function is a transcendental function defined by solutions of the equation $W(x)e^{W(x)} = x$. For real values of the argument, x, it has two branches, W_0 and W_{-1}, the principal and the negative branches [40].

To maintain the stochastic property of score vectors, we normalize columns of $\hat{\mathbf{R}}$ to get $\mathbf{R}$ such that

$$\mathbf{R}^{(t)}(i,j) = \begin{cases} \hat{\mathbf{R}}^{(t)}(i,j) / \sum_{i=1}^{S} \hat{\mathbf{R}}^{(t)}(i,j) & \text{if } \sum_{i=1}^{S} \hat{\mathbf{R}}^{(t)}(i,j) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

We reinforce each score vector $\mathbf{x}_i^{(t)}$ by adding positive effects from others. Specifically, the reinforced score vector is

$$\mathbb{x}_i^{(t)} = \begin{cases} (1-\gamma)\mathbf{x}_i^{(t)} + \gamma \sum_{j=1}^{S} \mathbf{R}^{(t)}(j,i)\mathbf{x}_j^{(t)} & \text{if } \sum_{j=1}^{S} \mathbf{R}^{(t)}(j,i) = 1 \\ \mathbf{x}_i^{(t)} & \text{if } \sum_{j=1}^{S} \mathbf{R}^{(t)}(j,i) = 0 \end{cases} \tag{5}$$

where γ ($0 \leq \gamma < 1$) is a factor to adjust the reinforcement effect. If $\gamma = 0$, there is no effect from other walkers.

At the next time point $(t+1)$, to compute vector $\mathbf{x}_i^{(t+1)}$, we set $\mathbf{x}_i^{(t)} \leftarrow \mathbb{x}_i^{(t)}$ and compute $\mathbf{x}_i^{(t+1)}$ based on Equation (2).

The proof of the convergence of $\mathbf{x}_i^{(t)}$ is more complicated after introducing reinforcements between the walkers. We omit the proof here due to the space limit. Interested readers are referred to Appendix C available in [1] for further details.

IV. Algorithm

In this section, we first present the algorithm for computing the score vectors $\mathbf{x}_i^{(t)}$. We then introduce several strategies to speed up the computation. Finally we discuss how to perform local clustering based on the reinforced score vectors.

A. Score Vector Computing

Algorithm 1 shows the overall process for computing $\mathbf{x}_i^{(t)}$ for all query nodes $q_i \in Q$. During the process, we use a table $keyP_{1,\ldots,S}$ (0,. . .,t) to store key positions from time point 0 to t, such that $keyP_i(t)$ stores key positions of $\mathbb{x}_i^{(t)}$. At time point $(t+1)$, for each query node q_i, the algorithm first updates the intermediate vector $\mathbf{x}_i^{(t+1)}$ based on Equation (2) (Line 6). Then we obtain the reinforcement matrix $\hat{\mathbf{R}}$ by computing the cosine similarity of each pair of $\mathbf{x}_i^{(t+1)}$ and $\mathbf{x}_j^{(t+1)}$ (Lines 7 to 12). Next we get the reinforced vector $\mathbb{x}_i^{(t+1)}$ by adding reinforced effects if others have positive affects on $\mathbf{x}_i^{(t+1)}$ (Lines 15 to 16). Otherwise we keep the score vectors unchanged (Line 17). After that we prepare the visiting history vector $\mathbf{v}_i^{(t+1)}$ for the next step at $(t+2)$ (Lines 18 to 20). In this process, we append $keyP_i$ with a new entry to store key positions of $\mathbb{x}_i^{(t+1)}$ (Line 18) and update vectors $\mathbf{e}_i^{(t+2-k)}(1 \leq k \leq K)$ based on $keyP_i$ (Line 19). Finally, we set $\mathbf{x}_i^{(t+1)} \leftarrow \mathbb{x}_i^{(t+1)}$ (Line 21) and check if it has converged (Line 22). The updating process will continue until all $\mathbf{x}_i^{(t)}(1 \leq i \leq S)$ converge. Then we obtain the final converged score vectors and the reinforcement matrix $\hat{\mathbf{R}}$.

Complexity Analysis: Given a graph $G(V,E)$ and a set of query nodes $Q = \{q_1, \ldots, q_S\}$, the time complexity of MRW in Algorithm 1 is $O(t_c S(|E| + S|V| + K))$, in which all score vectors $\mathbf{x}_i^{(t)}(1 \leq i \leq S)$ converge at t_c. At each time point,

Algorithm 1: Memory-based Random Walk (MRW)

Input: $\mathbf{P}$, α, β, γ, K, $Q = \{q_1, \ldots, q_S\}$, ϵ, θ
Output: $\mathbf{x}_i^{(t)}(1 \leq i \leq S)$, reinforcement matrix $\hat{\mathbf{R}}$

$t = 0$;
for $i = 1 : S$ **do**
  $\mathbf{x}_i^{(0)} = \mathbf{0}; \mathbf{x}_i^{(0)}(q_i) = 1$; $\mathbb{x}_i^{(0)} = \mathbf{x}_i^{(0)}$; $keyP_i(0) = q_i$;
  Init. uniform values for key posi. in $\mathbf{e}_i^{(0)}$ and $\mathbf{v}_i^{(0)}$;
while *any score vector* $\mathbf{x}_i$ *does not converge* **do**
  for $i = 1 : S$ **do** $\mathbf{x}_i^{(t+1)} = \alpha \mathbf{P}^\intercal \mathbf{x}_i^{(t)} + (1-\alpha)\mathbf{v}_i^{(t)}$;
  Initialize the reinforcement matrix $\hat{\mathbf{R}} = \mathbf{0}$;
  for $i = 1 : S$ **do**
    for $j = i + 1 : S$ **do**
      if $cos(\mathbf{x}_i^{(t+1)}, \mathbf{x}_j^{(t+1)}) > \theta$ **then**
        $\hat{\mathbf{R}}(i,j) = cos(\mathbf{x}_i^{(t+1)}, \mathbf{x}_j^{(t+1)})$;
        $\hat{\mathbf{R}}(j,i) = \hat{\mathbf{R}}(i,j)$;
  $\mathbf{R} \leftarrow$ Column-wise normalize $\hat{\mathbf{R}}$;
  for $i = 1 : S$ **do**
    if $\sum_{j=1}^{S} \mathbf{R}(j,i) == 1$ **then**
      $\mathbb{x}_i^{(t+1)} = (1-\gamma)\mathbf{x}_i^{(t+1)} + \gamma \sum_{j=1}^{S} \mathbf{R}(j,i)\mathbf{x}_j^{(t+1)}$;
    else $\mathbb{x}_i^{(t+1)} = \mathbf{x}_i^{(t+1)}$;
    Assign $keyP_i(t+1)$ with the key posi. of $\mathbb{x}_i^{(t+1)}$;
    Compute $\mathbf{e}_i^{(t+2-k)}(1 \leq k \leq K)$ based on $keyP_i$;
    Update $\mathbf{v}_i^{(t+1)}$ based on Equation (3);
    $\mathbf{x}_i^{(t+1)} \leftarrow \mathbb{x}_i^{(t+1)}$;
    Check whether $\mathbf{x}_i^{(t+1)}$ has converged;
  $t = t + 1$;
return $\mathbf{x}_i^{(t)}(1 \leq i \leq S)$, $\hat{\mathbf{R}}$;

updating their intermediate score vectors $\mathbf{x}_i^{(t)}(1 \leq i \leq S)$ needs $O(S|E|)$. The reinforcement matrix $\hat{\mathbf{R}}$ can be calculated with $O(S^2|V|)$. Adding reinforced effects needs $O(S^2|V|)$. Computing $\mathbf{e}_i^{(t+2-k)}(1 \leq k \leq K)$ and $\mathbf{v}_i^{(t+1)}$ to prepare next updating process costs $O(S|V| + SK)$. Checking convergence status costs $O(S|V|)$.

B. Speeding-up Strategies

In this section, we introduce two strategies to speed up the score vector computation.

Strategy 1. We can make the algorithm more efficient by merging highly similar score vectors. Specifically, at each time point t, according to the reinforcement matrix $\hat{\mathbf{R}}^{(t)}$, if the similarity between a pair of vectors $\mathbf{x}_i^{(t)}$ and $\mathbf{x}_j^{(t)}$ is high, e.g., above 0.8, we consider the two corresponding query nodes q_i and q_j are from the same community. We update $\mathbf{x}_i^{(t)}$ as the average of the two vectors, i.e., $\mathbf{x}_i^{(t)} \leftarrow (\mathbf{x}_i^{(t)} + \mathbf{x}_j^{(t)})/2$ and set $\hat{\mathbf{R}}^{(t)}(i,j) = \hat{\mathbf{R}}^{(t)}(j,i) = 1$. In the next steps, we only continue updating $\mathbf{x}_i^{(t)}$ but not $\mathbf{x}_j^{(t)}$.

Moreover, from Theorem 1, we know that for a single query node, its score vector will converge after time point t_c. To further reduce the cost associated with similarity computing, we can stop updating the similarity values after t_c steps.

Strategy 2. We can reduce the number of updating steps without losing much accuracy. Recall that Theorem 1 shows that for each query node, when $t > t_c = \lfloor \frac{\log(\epsilon|\beta-\alpha|/2)}{\log(\max\{\alpha,\beta\})} \rfloor$, the error of $\mathbf{x}_i^{(t)}$ (we omit the subscript "i" for simplicity in the following analysis) will be less than the tolerance ϵ. If we expand Equation (3), we can see that $\mathbf{v}^{(t)}$ is a linear combination of $\mathbf{e}^{(\tau)}(0 \leq \tau \leq t)$, i.e.,

$$\mathbf{v}^{(t)} = \sum_{\tau=0}^{t} \psi_\tau^{(t)}(\beta)\mathbf{e}^{(\tau)} \tag{6}$$

where $\psi_\tau^{(t)}(\beta)$ is the weight of $\mathbf{e}^{(\tau)}$. Moreover, we can verify that when $t > t_\beta = \log\epsilon / \log\beta$, $\psi_\tau^{(t)}(\beta)$ approaches to 0. (Interested readers are referred to Theorem 5 in Appendix A available in [1] for details.) This means that when $t > t_\beta$, $\mathbf{e}^{(t)}$ has no effect on $\mathbf{v}^{(t)}$, $\mathbf{x}^{(t)}$ and $\mathbb{x}^{(t)}$.

Theorem 2: For any query node $q_i \in Q$, when $t > t_\beta = \log\epsilon / \log\beta$, as $\epsilon \to 0$, $||\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t_\beta)}||_1 = O(\epsilon|\log\epsilon|)$ if $\alpha = \beta$; otherwise, $||\mathbf{x}_i^{(t)} - \mathbf{x}_i^{(t_\beta)}||_1 = O(\epsilon^{\min\{1, \log_\beta \alpha\}})$.

The proof is omitted here due to the space limit. The detailed proof can be found in Appendix B available in [1].

From Theorem 2, we can stop the algorithm earlier at time t_β instead of t_c without losing much accuracy.

C. Local Clustering

To detect local communities, for each vector (after merging), we first find nodes with the top-L largest scores. We set the default value of L to be 200 since most communities in real-world datasets are not very large [41]. Let $\{l_i\}(1 \leq i \leq L)$ represent the list of top-L nodes sorted in descending order. For each i $(1 \leq i \leq L)$, we compute the conductance [42] of the subgraph induced by node set $\{l_1, \ldots, l_i\}$. The node set with the smallest conductance will be returned as the target community.

Complexity Analysis: For each query node, generating the list of the top-L sorted nodes takes time $O(|V| + L\log L)$. Computing the conductances and finding the smallest one need $O(Ld_L)$, where d_L is the average degree of the top-L nodes.

V. Experimental Results

We conduct extensive experiments to evaluate the performance of the proposed method using a variety of real-world networks. All experiments are performed on a server with 64G memory, Intel Xeon 2.6GHz CPU, and Redhat OS. The code and datasets can be found in [1].

A. Datasets and State-of-the-Art Methods

Table II shows the statistics of the network datasets used in our experiments (AZ: Amazon; DB: DBLP; YT: YouTube; LJ: LiveJournal; OT: Orkut). The columns correspond to the number of nodes $|V|$, number of edges $|E|$, number of communities (#Com.), and the average and standard deviation of the community size. These datasets are provided with

TABLE II
STATISTICS OF THE REAL NETWORKS

	$\|V\|$	$\|E\|$	# Com.	Aver. Com. Size (std)[2]
AZ	334,863	925,872	75,149	15.06 (14.15)
DB	317,080	1,049,866	13,477	10.91 (20.49)
YT	1,134,890	2,987,624	8385	10.68 (19.56)
LJ	3,997,962	34,681,189	287,512	24.39 (25.66)
OT	3,072,441	117,185,083	6,288,363	188.50 (195.87)

ground-truth community labels and are publicly available at *http://snap.stanford.edu*.

We compare MRW with several state-of-the-art local community detection methods. PageRank-Nibble (PRN) [2] adopts the degree normalized random walk with restart (RWR) score to rank nodes and finds the local cluster that minimizes the external conductance. The heat kernel (HK) method [6] replaces the decay factors in PageRank-Nibble by the heat kernel parameters to find local clusters. The query-biased densest connected subgraph detection (QDC) method [5] assigns weights to nodes using RWR scores and finds the query biased densest subgraph. LEMON [22] finds the local cluster by seeking a sparse vector in a low-dimensional Krylov subspace spanned by the query vector. The multi-walker chain (MWC) method [8] uses an integrated team of walkers to capture the local community structure. MARW [7] uses multiple agents with a rope of fixed length to restrict them from travelling too far away from each other. The second-order random walk model (RWR^2) computes the second-order RWR scores with edge-transition probabilities to alleviate the query-bias issue of the memoryless first-order RWR model. *k*-core [33] and *k*-truss [30] are two algorithms based on optimizing certain structural measurements.

Note that all the methods discussed above aim to find a *single* target community. To compare MRW with these single target community detection methods, we degenerate MRW by removing the reinforcement process represented by Equation (5). We use MRW-sc (sc stands for single community) to represent this simplified version of MRW. Thus MRW-sc is only based on Equation (2) which can be applied to find a single target community for each query node at a time.

We also compare MRW with the methods that can handle multiple queries. OCSG [29] finds γ-quasi-k-clique components for the query nodes. MIS [39] and MDL [37] are connection subgraph finding methods. These two methods allow to find multiple connection subgraphs for a set of query nodes. As discussed in Section II, the goal of connection subgraph finding is to find a small number of nodes to connect the query nodes instead of the communities the query nodes belong to.

B. Accuracy Evaluation

In each trial, we randomly select 200 ground truth communities in each network and randomly select 4 nodes from each selected community as the query nodes. We apply the selected methods to detect local communities for these 200×4 query nodes. This process is repeated 100 times. We use *F1-score* to measure the accuracy of detected local communities. Given the discovered local community C' and the ground-truth community C, F1-score is defined as

$$F(C', C) = 2 \cdot \frac{\text{prec}(C', C) \times \text{rec}(C', C)}{\text{prec}(C', C) + \text{rec}(C', C)}$$

where $\text{prec}(C', C) = \frac{|C' \cap C|}{|C'|}$ is the precision and $\text{rec}(C', C) = \frac{|C' \cap C|}{|C|}$ is the recall.

The parameters of all methods are tuned to achieve their optimal performances. More detailed parameter sensitivity evaluations can be found in Section V-F.

Table III shows the average F1-score results of the selected methods. We divide these methods into two groups, i.e., the ones that allow multiple target communities/subgraphs and the ones that are designed to find a single target community. For the methods that allow multiple target communities, i.e., MRW, OCSG, MIS, and MDL, it is clear that MRW achieves the best performance. Note that both MIS and MDL aim to find connection subgraphs. Their returned subgraphs are usually small. MDL cannot run into completion after two days on the three larger datasets, YT, LJ, and OT, due to its pair-wise shortest path computation. For OCSG, the required γ-quasi-k-clique components are hard to obtain in certain datasets such as YT and OT. Most of the communities in these datasets are star structures [41].

Among the methods that find one target community for each query node, we can see that MRW-sc achieves the best results. The advantage of MRW-sc comes from the strategy that the walker memorizes its visiting history. This strategy can help the walker to effectively capture the local community structure instead of being biased to the query node. Other random walk-based methods, MWC, RWR^2, and QDC, and local spectral method, LEMON, also achieve reasonable results. The substructure optimization based methods, k-core and k-truss, usually cannot identify community structures that satisfy the definitions in their methods.

Furthermore, comparing the performance of MRW and MRW-sc in Table III, we can see that MRW performs better than MRW-sc. This demonstrates the effectiveness of the reinforcement process among the walkers. In the next section, we provide more detailed evaluation on the effectiveness of the reinforcement process.

C. Effectiveness of the Reinforcement

To further examine the effectiveness of the reinforcement process, we study how the performance of MRW will change with more query nodes taken from the same community.

From each of the 200 communities of each dataset in Section V-B, we randomly choose $\{2, 4, 6, 8\}$ distinct nodes as the query nodes. That is we use $\{200 \times 2, 200 \times 4, 200 \times 6, 200 \times 8\}$ query nodes in MRW. For each setting, we perform 100 trials and report the average result. Table IV shows the average F1-scores for different settings. We can see that MRW tends to

[2]The community size statistics are obtained based on the selected communities using the 3-sigma rule from the well structured top-5000 clusters of each network after deleting duplicated clusters [41].

TABLE III
F1-SCORES OF DETECTED COMMUNITIES FOR 200 × 4 QUERIES (200 RANDOM SELECTED COMMUNITIES AND 4 QUERIES FROM EACH COMMUNITY)

Methods	Multiple target communities				Single target community									
	MRW	OCSG	MIS	MDL	MRW-sc	MWC	MARW	RWR^2	QDC	HK	PRN	LEMON	k-core	k-truss
AZ	**0.9100**	0.7950	0.3696	0.4191	**0.8999**	0.8915	0.8246	0.8501	0.8213	0.5413	0.7834	0.8249	0.2937	0.7128
DB	**0.5301**	0.3586	0.2938	0.4213	**0.5223**	0.5056	0.4056	0.3703	0.4461	0.3058	0.4084	0.4314	0.1359	0.2192
YT	**0.2010**	0.0643	0.0234	–	**0.1884**	0.1864	0.0940	0.1046	0.1211	0.1089	0.0558	0.1519	0.0162	0.0859
LJ	**0.7090**	0.4202	0.1861	–	**0.7045**	0.7005	0.3716	0.6626	0.6400	0.4868	0.5406	0.6417	0.0833	0.4476
OT	**0.3609**	0.0350	0.0282	–	**0.3588**	0.3405	0.1428	0.2286	0.3019	0.1862	0.2452	0.2812	0.0111	0.0020
Aver.	**0.5422**	0.3346	0.1802	–	**0.5348**	0.5249	0.3677	0.4432	0.4661	0.3258	0.4067	0.4662	0.1080	0.2935

TABLE IV
F1-SCORES OF MRW FOR DIFFERENT NUMBER OF QUERIES

# Queries	200 × 2	200 × 4	200 × 6	200 × 8
AZ	0.9100	0.9100	0.9112	0.9127
DB	0.5012	0.5301	0.5513	0.5654
YT	0.1606	0.2010	0.2154	0.2421
LJ	0.6936	0.7090	0.7114	0.7212
OT	0.3525	0.3609	0.3628	0.3685
Avg.	0.5236	0.5422	0.5504	0.5620

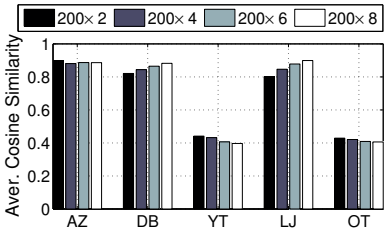
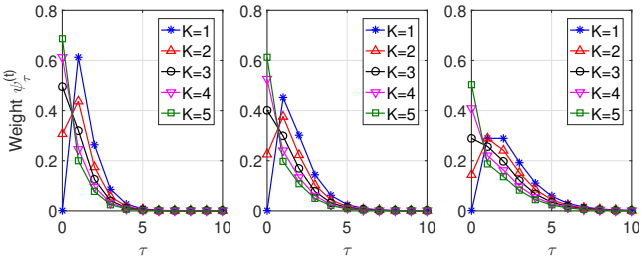


Fig. 2. The average cosine similarity between the visiting history score vectors in the same community



(a) $\beta = 0.3$ (b) $\beta = 0.4$ (c) $\beta = 0.5$

Fig. 3. Aggregated key position weight $\psi_\tau^{(t)}$ for different K and β ($t = 10$)

perform better when more query nodes are from the same community. This is due to the effectiveness of reinforcement process.

Next, we study the similarity values in the reinforcement matrix $\hat{\mathbf{R}}$. Figure 2 shows the average similarity value of score vectors corresponding to the queries from the same ground truth community. As we can see, for datasets AZ, DB and LJ, the similarities are very high. This is consistent with the accuracy results in Table IV. For datasets YT and OT, the average similarity value is around 0.4. The p-value of cosine similarity value 0.4 with 200 sample size (the typical size of sub-vectors based on which the similarity is calculated) is 10^{-4}, which is still statistically significant. This verifies that the walkers from the same community tend to have similar visiting histories.

D. Effectiveness of the Sliding Window

Equation (6) shows that $\mathbf{v}^{(t)}$ can be treated as the summarization of all previous key positions vectors $\mathbf{e}^{(\tau)}(0 \leq \tau \leq t)$. Figure 3 shows the distribution of weight $\psi_\tau^{(t)}$ of previous key positions for different β and K. The weight distribution in Figure 3 shows that MRW assigns weights to key positions with a focus on the neighborhood of the query. This is ideal since we want the walker to be seen in the neighbourhood of the query nodes. The existing memoryless methods (e.g., the widely used RWR) usually only assign a weight to the query node (e.g., $(1-\alpha)\mathbf{q}$ in Equation (1)) and do not consider the neighbourhood in later steps of the walker.

E. Efficiency Evaluation

In this section, we evaluate the efficiency of selected methods. The connection subgraph methods MIS and MDL are not included since they do not really find the target communities as shown in Table III.

Figure 4 shows the running time of different local community detection methods averaged on 800 query nodes. From the figure, we can see that MRW and MRW-sc are always among the top-3 fastest methods. PRN and HK are fast because of their "push" strategy that is used to approximate the scores. LEMON's fast speed comes from its sampling process which first gets a subgraph around the query node, and then applies spectral clustering in the small subgraph. However, these methods do not provide competitive accuracy results. Optimization-based algorithms, OCSG, k-core and k-truss, are slow since they need to recursively find subgraphs satisfying their cohesiveness definitions. The methods that focus on complex models, such as MWC and MARW, which use multiple walkers or agents for each query node, and RWR^2, which is based on the dense second-order transition matrix, are usually computationally expensive. MRW is much more efficient than these methods. In MRW, only one walker is associated with each query node. The score vectors of walkers

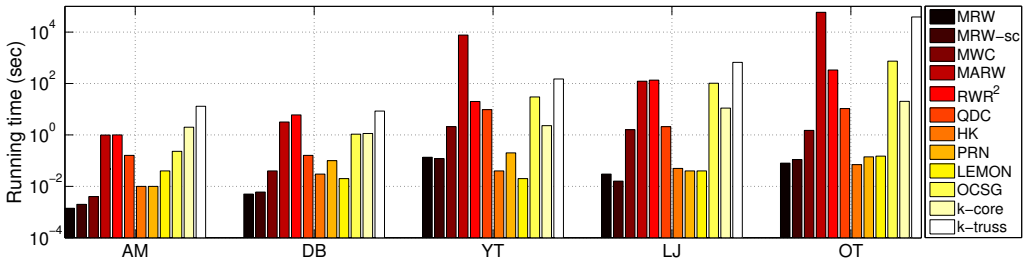
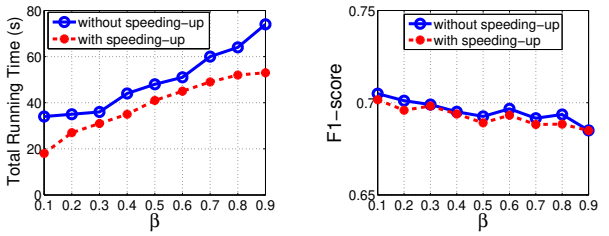


Fig. 4. Average running time over all query nodes



(a) Running time (b) F1-score

Fig. 5. Speeding-up strategy evaluation

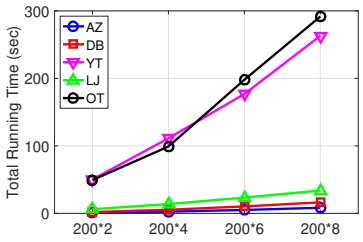
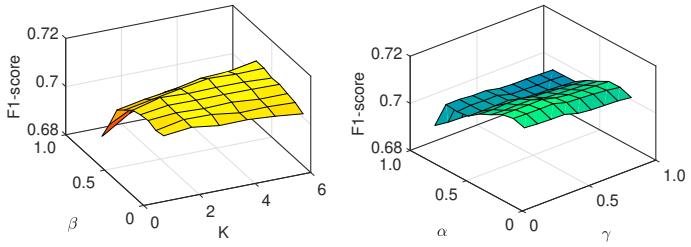


Fig. 6. Running time of MRW with different number of query nodes



(a) β vs. K (b) α vs. γ

Fig. 7. Sensitivity evaluation on LJ

can be merged during the reinforcement process thus further reducing the computing time.

Figures 5(a) and 5(b) show the running time and average F1-score of MRW for the 200×4 query nodes on LJ dataset (as discussed in Section V-C) with and without speeding-up strategies by tuning β. From Figure 5(a), we can see that smaller β can shorten the convergence time which verifies the theoretical analysis in Sections III-B and IV-B. In addition, the speeding-up strategies can save computing time without significant accuracy loss as shown in Figure 5(b).

Figure 6 shows the running time of MRW for different numbers of query nodes. Based on the complexity analysis in Section IV-A, since the network size is usually much larger than the size of input query node set, i.e., $|V| >> S$, in practice, the running time of MRW would be roughly linear with respect to S. Figure 6 verifies this observation.

F. Sensitivity Evaluation

Figure 7 shows the effect of α, β, γ and K on F1-score in LJ dataset (similar trends occur in other datasets). We can observe that when $0.3 \leq \beta \leq 0.6$, MRW achieves the best performance. Increasing K gives slightly better results. Intuitively, larger K means that the current sliding window memorizes more previous steps. A smaller α value gives better performance. This is also intuitive since a smaller α value indicates that visiting history plays a more important role in deciding next steps of the walker. γ balances the walkers own memory history and the influence from others. When $0.2 \leq \gamma \leq 0.4$, MRW achieves the best.

G. Case Study

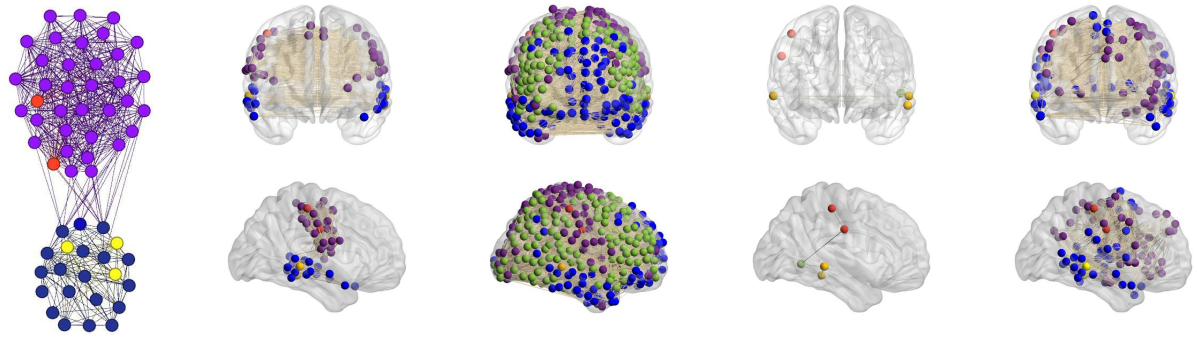
In this section, we apply MRW to two interesting real-world networks, the human brain network and the flavor compound network, to show its effectiveness in the real-world applications.

1) Human Brain Network: Network analysis has been increasingly used in human brain studies [43]. In our case study, we apply MRW in a human brain co-activation network [44] and show that the detected communities have spatial and functional meanings.

The brain network in [44] has 638 nodes which correspond to the cortical areas of the human brain and 18625 edges which measure the functional associations between the cortical areas. The 3D coordinates of the nodes in the human brain are also provided. Based on their coordinates, we can map the nodes into the 52 Brodmann areas (e.g., using the Talairach Client[3]) with known functions.

We randomly select five nodes from two Brodmann areas. Two query nodes are from area 2 and the other three are from

[3]http://www.talairach.org/



(a) MRW topology (b) MRW (c) OCSG (d) MIS (e) MWC

Fig. 8. Local communities detected in the brain co-activation network. The query nodes are labelled in red and yellow.

area 21. The functionalities associated with these two areas are somatosensory and language, respectively[4].

The local communities detected by MRW for the five query nodes in the brain co-activation network are shown in Figure 8(a). The two query nodes in area 2 are colored in red and the three query nodes in area 21 are in yellow. From the figure, we can see that MRW automatically detect the two communities (represented by purple nodes and blue nodes, respectively) that the five query nodes belong to. We further map the nodes in the detected communities to the 3D brain cortical surface using BrainNet Viewer [45]. The result is shown in Figure 8(b). The two communities are clearly separated. In terms of the functionality, most of the purple nodes are associated with somatosensory (corresponding to area 2) and most of the blue nodes are associated with language (corresponding to area 21). These results show that MRW successfully detects the relevant brain regions that the query nodes belong to.

We also show the results of other methods in Figures 8(c)-8(e). We can see that OCSG (Figure 8(c)) includes almost all the nodes in the detected communities (the green nodes are overlapping nodes of two communities). MIS (Figure 8(d)) adds only one more node in the right hemisphere in the result. Figure 8(e) shows the result of MWC, which is applied to find community for each query node and then merge highly overlapping ones. The detected communities contain many nodes corresponding to functionalities that are irrelevant to the target communities.

2) Flavor Compound Network: Culinary practice plays an important role in human history. Nowadays people pay much attention to the flavor compound profiles of culinary ingredients to build personalized diet. The flavor compound network which represents the compound similarity between different ingredients can be used for this purpose [46]. Nodes in the flavor compound network represent ingredients. There is an edge between two ingredients if they share similar flavor compounds. One can build new recipes from existing ones according to the detected communities in the flavor network for the key ingredients. For instance, in our case study, for

[4]The functionalities of the Brodmann areas can be obtained from http://www.fmriconsulting.com/brodmann/Interact.html

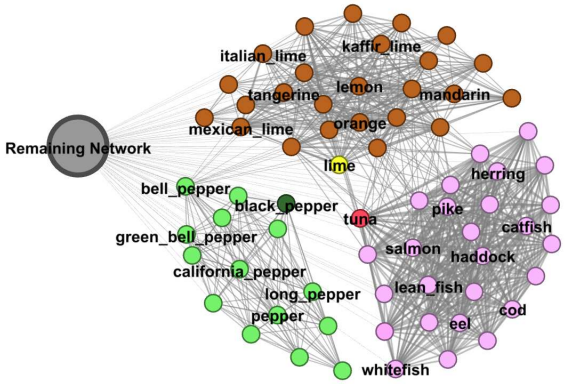


Fig. 9. Local communities detected in the flavor compound network. The query ingredients are tuna, black pepper, and lime, which are colored in red, dark green, and yellow respectively.

a common recipe *Tuna with Lime and Black Pepper Sauce*, we use its three main ingredients, i.e., tuna, black pepper, and lime, as the query nodes in the flavor compound network. We apply MRW to detect their corresponding communities to identify potential ingredient replacements that contain similar flavor compounds.

The detected communities are shown in Figure 9. The query ingredients, tuna, black pepper, and lime, are colored in red, dark green, and yellow, respectively. MRW identifies three communities that the three query ingredients belong to. We find that the ingredients in each community do share similar compounds with their corresponding query ingredient. For query ingredient tuna, its community members are various kinds of fish. For query black pepper, its community members are different peppers and cayenne. For query lime, most its community members are orange related ingredients. Based on the detected communities, we may build another dish such as *Salmon With Lemon and Bell Pepper Saute* providing similar flavor compounds with the tuna recipe.

We also applied other methods on the flavor compound network and found that their results are similar to the ones in the brain network. For example, OSCG returns very large communities while MIS returns small ones. MWC includes

irrelevant nodes in the identified community.

VI. Conclusion

In this paper, we propose a memory-based random walk model, MRW, to automatically identify target communities for multiple query nodes. The existing methods assume that all query nodes are from the same community. This assumption is not true in many real-world applications. In MRW, we assign a walker to each query node and use a sliding window approach to aggregate the visiting history for each walker. The walkers coming from the same community can mutually reinforce each other during the process. We devise efficient algorithms to compute the visiting probability score vectors and provide rigorous theoretical foundation of the proposed method. Extensive experimental evaluations on various real-world datasets demonstrate the effectiveness and efficiency of the proposed MRW model.

Acknowledgement

This work was partially supported by the National Science Foundation grants IIS-1664629 and CAREER.

References

[1] Supplementary, *http://sites.psu.edu/yuchenbian/mrw/*.
[2] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *FOCS*, 2006.
[3] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning," *SIAM Journal on Computing*, vol. 42, no. 1, pp. 1–26, 2013.
[4] R. Andersen and K. J. Lang, "Communities from seed sets," in *WWW*, 2006.
[5] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: on free rider effect and its elimination," in *VLDB*, 2015.
[6] K. Kloster and D. F. Gleich, "Heat kernel based community detection," in *KDD*, 2014.
[7] M. Alamgir and U. Von Luxburg, "Multi-agent random walks for local clustering on graphs," in *ICDM*, 2010.
[8] Y. Bian, J. Ni, W. Cheng, and X. Zhang, "Many heads are better than one: Local community detection by the multi-walker chain," in *ICDM*, 2017.
[9] I. M. Kloumann and J. M. Kleinberg, "Community membership identification from small seed sets," in *KDD*, 2014.
[10] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using seed set expansion," in *CIKM*, 2013.
[11] S. E. Baranzini, N. W. Galwey, J. Wang, and et.al., "Pathway and network-based analysis of genome-wide association studies in multiple sclerosis," *Human molecular genetics*, vol. 18, no. 11, pp. 2078–2090, 2009.
[12] K. Wang, M. Li, and H. Hakonarson, "Analysing biological pathways in genome-wide association studies," *Nature Reviews Genetics*, vol. 11, no. 12, p. 843, 2010.
[13] D.-Y. Cho, Y.-A. Kim, and T. M. Przytycka, "Network biology approach to complex diseases," *PLoS computational biology*, vol. 8, no. 12, p. e1002820, 2012.
[14] H. C. Akakin and M. N. Gurcan, "Content-based microscopic image retrieval system for multi-image queries," *IEEE transactions on information technology in biomedicine*, vol. 16, no. 4, pp. 758–769, 2012.
[15] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, 2001.
[16] Y. Wu, Y. Bian, and X. Zhang, "Remember where you came from: On the second-order random walk based proximity measures," in *VLDB*, 2016.
[17] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *KDD*, 2017.
[18] D. Zhou, S. Zhang, M. Y. Yildirim, S. Alcorn, H. Tong, H. Davulcu, and J. He, "A local algorithm for structure-preserving graph cut," in *KDD*, 2017.
[19] C. Lee, W.-D. Jang, J.-Y. Sim, and C.-S. Kim, "Multiple random walkers and their application to image cosegmentation," in *CVPR*, 2015.
[20] S.-H. Lee, W.-D. Jang, B. K. Park, and C.-S. Kim, "Rgb-d image segmentation based on multiple random walkers," in *ICIP*, 2016.
[21] K. He, Y. Sun, D. Bindel, J. Hopcroft, and Y. Li, "Detecting overlapping communities from local spectral subspaces," in *ICDM*, 2015.
[22] Y. Li, K. He, D. Bindel, and J. E. Hopcroft, "Uncovering the small community structure in large networks: A local spectral approach," in *WWW*, 2015.
[23] K. He, P. Shi, J. E. Hopcroft, and D. Bindel, "Local spectral diffusion for robust community detection," in *Twelfth Workshop on Mining and Learning with Graphs*, 2016.
[24] A. R. Benson, D. F. Gleich, and J. Leskovec, "Tensor spectral clustering for partitioning higher-order network structures," in *SDM*, 2015.
[25] J. P. Bagrow and E. M. Bollt, "Local method for detecting communities," *Physical Review E*, vol. 72, no. 4, p. 046108, 2005.
[26] A. Clauset, "Finding local community structure in networks," *Physical review E*, vol. 72, no. 2, p. 026132, 2005.
[27] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.
[28] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based densest clique percolation community search in networks," *TKDE*, vol. 30, no. 5, pp. 922–935, 2018.
[29] J. Shan, D. Shen, T. Nie, Y. Kou, and G. Yu, "Searching overlapping communities for group query," *World Wide Web*, vol. 19, no. 6, pp. 1179–1202, 2016.
[30] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, 2014.
[31] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proceedings of the VLDB Endowment*, vol. 9, no. 4, pp. 276–287, 2015.
[32] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309.
[33] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data Mining and Knowledge Discovery*, vol. 29, no. 5, pp. 1406–1433, 2015.
[34] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *Proceedings of the VLDB Endowment*, vol. 10, no. 6, pp. 709–720, 2017.
[35] X. Huang, L. V. Lakshmanan, and J. Xu, "Community search over big graphs: Models, algorithms, and opportunities," in *ICDE*, 2017.
[36] H. Tong and C. Faloutsos, "Center-piece subgraphs: problem definition and fast solutions," in *KDD*, 2006.
[37] L. Akoglu, D. H. Chau, J. Vreeken, N. Tatti, H. Tong, and C. Faloutsos, "Mining connection pathways for marked nodes in large graphs," in *SDM*, 2013.
[38] N. Ruchansky, F. Bonchi, D. García-Soriano, F. Gullo, and N. Kourtellis, "The minimum wiener connector problem," in *SIGMOD*, 2015.
[39] N. Ruchansky, F. Bonchi, D. Garcia-Soriano, F. Gullo, and N. Kourtellis, "To be connected, or not to be connected: That is the minimum inefficiency subgraph problem," in *CIKM*, 2017.
[40] D. Barry, J.-Y. Parlange, L. Li, H. Prommer, C. Cunningham, and F. Stagnitti, "Analytical approximations for real values of the lambert w-function," *Mathematics and Computers in Simulation*, vol. 53, no. 1, pp. 95–103, 2000.
[41] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *ICDM*, 2012.
[42] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.
[43] X. Kong and P. S. Yu, "Brain network analysis: a data mining perspective," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 2, pp. 30–38, 2014.
[44] N. A. Crossley, A. Mechelli, P. E. Vértes, T. T. Winton-Brown, A. X. Patel, C. E. Ginestet, P. McGuire, and E. T. Bullmore, "Cognitive relevance of the community structure of the human brain functional coactivation network," vol. 110, no. 28, 2013, pp. 11 583–11 588.
[45] M. Xia, J. Wang, and Y. He, "Brainnet viewer: a network visualization tool for human brain connectomics," *PloS one*, vol. 8, no. 7, p. e68910, 2013.
[46] Y.-Y. Ahn, S. E. Ahnert, J. P. Bagrow, and A.-L. Barabási, "Flavor network and the principles of food pairing," *Scientific reports*, vol. 1, p. 196, 2011.