Journal of Big Data

RESEARCH Open Access

Limited random walk algorithm for big graph data clustering

Honglei Zhang[1*], Jenni Raitoharju[1], Serkan Kiranyaz[2] and Moncef Gabbouj[1]

*Correspondence: honglei.zhang@tut.fi
[1] Department of Signal Processing, Tampere University of Technology, Korkeakoulunkatu 1, 33101 Tampere, Finland
Full list of author information is available at the end of the article

Abstract
Graph clustering is an important technique to understand the relationships between the vertices in a big graph. In this paper, we propose a novel random-walk-based graph clustering method. The proposed method restricts the reach of the walking agent using an inflation function and a normalization function. We analyze the behavior of the limited random walk procedure and propose a novel algorithm for both global and local graph clustering problems. Previous random-walk-based algorithms depend on the chosen fitness function to find the clusters around a seed vertex. The proposed algorithm tackles the problem in an entirely different manner. We use the limited random walk procedure to find attractor vertices in a graph and use them as features to cluster the vertices. According to the experimental results on the simulated graph data and the real-world big graph data, the proposed method is superior to the state-of-the-art methods in solving graph clustering problems. Since the proposed method uses the embarrassingly parallel paradigm, it can be efficiently implemented and embedded in any parallel computing environment such as a MapReduce framework. Given enough computing resources, we are capable of clustering graphs with millions of vertices and hundreds millions of edges in a reasonable time.

Keywords: Graph clustering, Random walk, Big data, Community finding

Background

Graph data are important data types in many scientific areas, such as social network analysis, bioinformatics, and computer and information network analysis [1]. In recent years, the size of graph data has grown dramatically. For example, a typical social network graph may contains millions of vertices and hundreds of million of edges. Further more, these graphs may continuously evolve over time. Processing these dynamic big graph data is very challenging and time consuming. In general, big graphs are normally heterogeneous. They have such non-uniform structures that edges between vertices in a group are much denser than edges connecting vertices in different groups. Graph clustering (also named as "community detection" in the literature) algorithms aim to reveal the heterogeneity and find the underlying relations between vertices [2]. This technique is critical for understanding the properties, predicting dynamic behavior and improving visualization of big graph data.

Graph clustering is a computationally challenging and difficult task, especially for big graph data. Many algorithms have been proposed over the last decades [2–4]. The

© The Author(s) 2016. This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

criteria-based approaches try to optimize clustering fitness functions using different optimization techniques. Newman defined a modularity measurement based on the probability of the link between any two vertices. He applied a greedy search method to minimize this modularity fitness function in order to partition a graph into clusters [5]. Blondel et al. used the same fitness function but combined it with other optimization techniques [6–8]. Spielman and Teng opted the graph conductance measurement as the fitness function [9]. Other than criteria-based methods, spectral analysis has also been widely adapted in this area [10, 11]. Random-walk-based methods tackle the problem from a different angle [12–14]. These methods use the Markov chain model to analyze the graph. Each vertex represents a state and the edges indicate transitions between the states. The probability values that are distributed among the states (vertices) reveal the graph structure.

For big graph data, the problem becomes more challenging or even intractable. Very often, people are only interested in finding the cluster for a given seed vertex. This problem is called local clustering problem [9, 15, 16]. For example, from an end user's perspective, finding the closely connected friends around him or her is more important than revealing the global user clusters of a large social network. It is unnecessary to explore the whole graph structure for this problem. Recently, random walk methods have gained great attention on this local graph clustering problem, since a walk started from the seed vertex is more likely to stay in the cluster where the seed vertex belongs. Comparing to the criteria-based methods, the random-walk-based methods are capable of extracting local information from a big graph without the knowledge of the whole graph data. In [17–19], a random walk is first applied to find important vertices around the seed vertex. Then a sweep stage is involved to select the vertices that minimize the conductance of the candidate clusters.

The accuracy of any criteria-based clustering method (or those combined with the random walk procedures) is greatly affected by the chosen clustering fitness function. Furthermore, most local clustering algorithms use the criteria that are more suitable for the global graph clustering problem. These choices greatly degrade the performance of these algorithms when the graph is big and highly uneven. Also the majority of the graph clustering algorithms are designed in sequential computing paradigm. Therefore, they do not take advantage of modern high-performance computing systems.

In this paper, we propose a novel random-walk-based graph clustering algorithm—the limited random walk (LRW) algorithm. First of all, the LRW algorithm does not rely on any clustering fitness function. Furthermore, the proposed method can efficiently tackle the computational complexity using a parallel programming paradigm. Finally, as a unique property among many graph clustering methods, LRW can be adapted to both global and local graph clustering in an efficient way.

The rest of the paper is organized as follows: basics of random walk procedure and the proposed LRW algorithm are explained in "Methodology" section; an extensive set of experiments on the simulated and real graph data, along with both numerical and visual evaluations are given in "Experiments" section; finally, the conclusions and future work are discussed in "Conclusions" section.

Methodology

Basic definitions and the random walk procedure

Let $G(V, E)$ denote a graph of n vertices and m edges, where $V = \{v_i | i = 1, \dots n\}$ is the set of vertices and $E = \{e_i | i = 1, \dots m\}$ is the set of edges. Let $A \in \mathcal{R}^{n \times n}$ be the adjacency matrix of the graph G and A_{ij} are the elements in the matrix A. Let $D \in \mathcal{R}^{n \times n}$ be the degree matrix, which is a diagonal matrix whose elements on the diagonal are the degrees of each vertex. In this paper, we assume the graph is undirected, unweighted and does not contain self-loops.

Clustering phenomenon is very common in big graph data. A cluster in a graph is a vertex set where the density of the edges inside the cluster is much higher than the density of edges that link the inside vertices and the outside vertices.

Random walk on a graph is a simple stochastic procedure. At the initial state, an agent stays on a chosen vertex (seed vertex). At each step, the agent randomly picks a neighboring vertex and moves to it. The agent repeats this movement and there is certain probability that the agent lands on a vertex after each movement.

Let $x_i^{(t)}$ denote the probability that the agent is on vertex v_i after step t, where $i = 1, 2, \dots n$. $x_i^{(0)}$ is the probability of the initial state. Let s be the seed vertex. We have $x_s^{(0)} = 1$, and $x_i^{(0)} = 0$ for $i \neq s$. Let $x^{(t)} = \left[x_1^{(t)}, x_2^{(t)}, \dots, x_n^{(t)}\right]^T$ be the probability vector, where the superscript T denotes the transpose of a matrix or a vector. By the definition of the probability, it is easy to see that $\sum_{i=1}^{n} x_i^{(t)} = 1$ or $\left\|x^{(t)}\right\|_1 = 1$.

The random walk procedure is equivalent to a discrete-time stationary Markov chain process. Each vertex is corresponding to a state in the Markov chain and each edge indicates a possible transition between the two states. The Markov transition matrix P can be obtained by normalizing the adjacency matrix to have each column sum up to 1, e.g.

$$P_{ij} = \frac{A_{ij}}{\sum_{k=1}^{n} A_{kj}} \tag{1}$$

or

$$P = AD^{-1}. \tag{2}$$

Other forms of the transition matrix P can also be used, for example the lazy random walk uses transition matrix $P = \frac{1}{2}(I + AD^{-1})$, where I is the identity matrix. Given the transition matrix P, we can calculate $x^{(t+1)}$ from $x^{(t)}$ using the equation:

$$x^{(t+1)} = Px^{(t)}. \tag{3}$$

A closed walk is a walk on a graph where the ending vertex is same as the seed vertex. The period of a vertex is defined as the greatest common divisor of the lengths of all closed walks that start from this vertex. We say a graph is aperiodic if all of its vertices have periods of 1.

For an undirected, connected and aperiodic graph, there exists an equilibrium state π, such that $\pi = P\pi$. This state is unique and irrelevant to the starting point. By iterating Eq. 3, $x^{(t)}$ converges to π. More details about the Markov chain process and the equilibrium state can be found from [20].

Limited random walk procedure

Definitions

We first define the transition matrix P. We assign the same probability to the transition that the walking agent stays in the current vertex and the transition that it moves to any neighboring vertex. We add an identity matrix to the adjacency matrix and then normalize the result to have each column sum to 1. The transition matrix can be written as

$$P = (I + A)(I + D)^{-1}. \tag{4}$$

Comparing to the transition matrix in Eq. 2, this is similar to adding self-loops to the graph, but increasing the degree of each vertex by 1 instead of 2. This modification fixes the periodicity problem that the graph may have [20]. It greatly improves the stability and accuracy of the algorithm in graph clustering.

At each walking step, the probability vector $x^{(t)}$ is computed using Eq. 3. Note that, in general, elements in $x^{(t)}$ that are around the seed vertex are non-zeros and the rest are zeros. So we do not need the full transition matrix to calculate the probability vector for the next step.

Starting from the seed vertex, a normal random walk procedure will eventually explore the whole graph. To reveal a local graph structure, different techniques can be used to limit the scope of the walks. Harel and Koren fix the number of walking steps by a predefined constant [21]. Xin et al. use a stochastic method to determine if a walk should be continued and set the maximum number of walking steps to be 6 according to the principle of "six degrees of separation" [14]. In [13, 17, 22], the random walk function is defined as

$$x^{(t+1)} = \alpha x^{(0)} + (1 - \alpha) P x^{(t)}, \tag{5}$$

where α is called the teleport probability. The idea is that there is a certain probability that the walking agent will teleport back to the seed vertex and continue walking.

Inspired by the Markov clustering algorithm (MCL) algorithm [12], we adapt the inflation and normalization operation after each step of the transition. The inflation operation is an element-wise super-linear function—a function that grows faster than a linear function. Here we use the power function

$$f(x) = \left[x_1^r, x_2^r, \ldots, x_n^r\right]^T, \tag{6}$$

where the exponent $r > 1$. Since x indicates the probability that the agent hits each vertex, x must be normalized to have a sum of 1 after the inflation operation. The normalization function is defined as

$$g(x) = \frac{x}{\|x\|_1}, \tag{7}$$

where $\|x\|_1 = \sum_{i=1}^{n} |x_i|$ is the L_1 norm of the vector x. Since $x_i \geq 0$ and $\sum_{i=1}^{n} x_i = 1$, Eq. 7 can also be written in a vector form as

$$g(x) = \frac{x}{x^T \cdot \mathbf{1}}, \tag{8}$$

where $\mathbf{1} = [1, 1, \ldots 1]^T$. The inflation and normalization operation enhance large values and depress small values in the vector x.

We call the aforementioned procedure the limited random walk (LRW) procedure. Comparing to the normal random walk procedure defined in "Basic definitions and the random walk procedure" section, LRW involves inflation and normalization operations in each walking step. These nonlinear operations limit the agent to walk around the neighborhood of the seed vertex, especially if there is a clear graph cluster boundary.

The MCL algorithm simulates flow within a graph. It uses the inflation and normalization operation to enhance the flow within a cluster and reduce the flow between clusters. The MCL procedure is a time-inhomogeneous Markov Chain in which the transition matrix varies over time. The MCL algorithm starts the random walk from all vertices simultaneously—there are n agents walking on the graph at the same time. The walking can only continue after all agents have completed a walking step and the result probability matrix has been inflated and normalized. Unlike in the MCL algorithm, the LRW procedure is a time-homogeneous Markov Chain. We initiate random walk from a single seed vertex, and do the inflation on the probability values of this walking agent. This design has many advantages. First, it avoids unnecessary walks since the graph structure around the seed vertex may be exposed by a single walk. Second, the procedure is suitable for the local clustering problems because it does not require the whole graph data. Third, if multiple walks are required, each walk procedure can be executed independently. Thus the algorithm is fully parallelizable.

The LRW procedure involves a nonlinear operation, thus it is difficult to analyze its properties on a general graph model. Next we study the equilibrium of the LRW procedure.

Equilibrium of the LRW procedure

We first prove the existence of equilibrium of the LRW procedure. Let X be the set of values of the probability vector x. We have

$$X = \{(x_1, x_2, \ldots, x_n) \mid 0 \le x_1, x_2, \ldots, x_n \le 1 \text{ and } x_1 + x_2 + \cdots x_n = 1\}. \tag{9}$$

The LRW procedure defined by Eqs. 3, 6 and 7 is a function that maps X to itself. Let $\mathcal{L} : X \to X$, such that

$$\mathcal{L}(x) = g(f(Px)). \tag{10}$$

Theorem 1 *There exists a fixed-point x^* such that $\mathcal{L}(x^*) = x^*$.*

Proof We use the Brouwer fixed-point theorem to prove this statement.

Given $0 \le x_1, x_2, \cdots, x_n \le 1$, the set X is clearly bounded and closed. Thus X is a compact set.

Let $u, v \in X$ and $w = \lambda u + (1-\lambda)v$, where $\lambda \in \mathcal{R}$ and $0 \le \lambda \le 1$. So $w_i = \lambda u_i + (1-\lambda)v_i$ for $i = 1, 2, \cdots n$. Obviously $0 \le w_i \le 1$.

Further,

$$\begin{aligned}\sum_{i=1}^{n} w_i &= \sum_{i=1}^{n} (\lambda u_i + (1-\lambda)v_i) \\ &= \lambda \sum_{i=1}^{n} u_i + (1-\lambda) \sum_{i=1}^{n} v_i \\ &= 1\end{aligned}$$

Thus $w \in X$. This indicates that the set X is convex.

Since function $f(x)$ is continuous over the set X and function $g(x)$ is continuous over the codomain of function $f(x)$, function $\mathcal{L}$ is continuous over the set X.

Given $\mathcal{L}$ is a continuous function that maps a convex set to itself, according to the Brouwer fixed-point theorem, there is a point x^* such that $\mathcal{L}(x^*) = x^*$. □

Theorem 1 shows the existence of fixed-point of the LRW procedure, i.e., the LRW procedure will not escape from a fixed-point whenever the point is reached. Since the LRW procedure is a non-linear discrete dynamic system, it is difficult to analytically investigate the system behavior. However, when $r = 1$, the LRW procedure is simply a Markov chain process, in which the fixed-point x^* is the unique equilibrium state π and the global attractor. In another extreme case when $r \to \infty$, a fixed-point can be an unstable equilibrium and the LRW procedure may have limit cycles that oscillate around a star structure in the graph. In one state of the oscillation, the probability value of the center of a star structure is close to one. In practice, we chose r from (1, 2]. This makes the LRW procedure close to a linear system and oscillations are extremely rare. In this case, the fixed-points of the LRW procedure are stable equilibriums.

Limited random walk on general graphs

Without any prior knowledge of the cluster formation, we normally start the LRW procedure from an initial state where $x_s = 1$, $x_i = 0$ for $i \neq s$ and s is the seed vertex. During the LRW procedure, there are two simultaneous processes—the spreading process and the contracting process. When the two processes can balance each other, a stationary state is reached.

During the spreading process, the probability values spread as the walking agent visits new vertices. The number of visited vertices increases exponentially at first. The growth rate depends on the average degree of the graph. The newly visited vertices will always receive the smallest probability values. If the graph has an average degree of d, it is not difficult to see that the expected probability value of a newly visited vertex at step t is $(1/d)^t$. As the walking continues, the probability values tend to be distributed more evenly among all visited vertices.

The other ongoing process during the LRW procedure is the contracting process. During this process, the probability values of the visited vertices contract to some vertices. Since the graph is usually heterogeneous, some vertices (and groups of vertices) will receive higher probability values as the procedure continues. The inflation operation further enhances this contracting effect. The degrees of a vertex and its surrounding vertices determine whether the probability values concentrate to or diffuse from these vertices. Some vertices, normally the center of a star structure, receive larger probability values than others. We call these vertices attractor vertices and they can be used to represent the structure of a graph.

Because the density of edges inside a cluster is higher than that of linking the vertices inside and outside the cluster, the probability that a walking agent visits vertices outside the cluster is small. Thus, the LRW procedure will find attractor vertices that the seed vertex is associated. We can use these vertices as features to cluster the vertices.

The larger the inflation exponent r is, the faster the algorithm converges to the attractor vertices. The LRW procedure tries to find the attractor vertices that are near the seed vertex. However, if r is too large, the probability values concentrate to the nearest attractor vertex (or the seed vertex itself) before the graph is sufficiently explored. If *r* is too small, the probability values will concentrate to the attractor vertices that may belong to other clusters. The performance of the LRW algorithm depends on choosing a proper inflation exponent *r*. From this aspect, it is similar to the MCL algorithm. In practice, *r* is normally chosen between 1 and 2 and the value 2 was found to be suitable for most graphs.

LRW for global graph clustering problems

In this section, we propose how to LRW in global graph clustering problems. Our algorithm is divided into two phases—graph exploring phase and cluster merging phase. To improve the performance on big graph data, we also propose a multi-stage strategy.

Graph exploring phase

In the graph exploring phase, the LRW procedure is started from several seed vertices. At each iteration, the agent moves one step as defined in Eq. 3. Then the probability vector x is inflated by Eq. 6 and normalized by Eq. 7. The iteration stops when the probability vector x converges or the predefined maximum number of iterations is reached. Let $x^{(*,i)}$ denote the final probability vector of a random walk that was started from the seed vertex v_i. As described in the previous section, the LRW procedure explores the vertices that are close to the seed vertex. Thus, the vector $x^{(*,i)}$ has non-zero elements only on these neighboring vertices.

Algorithm 1 illustrates the graph exploring from a seed vertex set Q. Note that for small graph data, we can set the seed vertex set $Q = V$ (i.e. the whole graph). In such case, the LRW procedure is executed on every vertex of the graph and the multi-stage strategy is not used.

Algorithm 1 Limited random walk graph exploring from a vertex set

```
given adjacency matrix $A$ of the graph $G(V,E)$, the seed vertex set $Q$, the exponent $r$ of the inflation
function 6, maximum number of iterations $T_{max}$, a small value $\epsilon$ to limit the number of visited
vertices, and a small value $\xi$ for the termination condition
initialize the transition matrix $P$ by Eq. 4
for each vertex $i$ in the vertex set $Q$
    1  initialize $x^{(0,i)}$ such that $x_k^{(0,i)} = \begin{cases} 1 & k = i \\ 0 & \text{otherwise} \end{cases}$ and $t = 1$
    2  repeat if $t < T_{max}$
         (a) $x^{(t,i)} = Px^{(t-1,i)}$
         (b) if $x_k^{(t,i)} < \epsilon$, set $x_k^{(t,i)}$ to 0
         (c) $x_k^{(t,i)} = f(x_k^{(t,i)}) = \quad x_k^{(t,i)} \quad {}^r$
         (d) $x_k^{(t,i)} = \frac{x_k^{(t,i)}}{\|x^{(t,i)}\|_1}$
         (e) if $\quad x^{(t,i)} - x^{(t-1,i)} \quad {}_2 < \xi$, exit the loop
         (f) $t \leftarrow t + 1$
    3  output $x^{(t,i)}$ as the attractor vector $x^{(*,i)}$ for vertex $i$
```

Note that the threshold ϵ limits the number of nonzero elements in the probability vector x. It is easy to prove that the number of nonzero elements in $x^{(t,i)}$ is less than $1/\epsilon$. A larger ϵ eliminates very small values in $x^{(t,i)}$ and prevent unnecessary computing efforts. However, ϵ does not impose a limit on the largest cluster we can find. Further, the choice of ϵ has little impact on the final clustering results because either the LRW procedure finds the most dominant attractor vertices in a cluster or the small clusters are merged in the cluster merging phase.

Cluster merging phase

After the graph has been explored, we will find the clusters in the cluster merging phase. We treat each $x^{(*,i)}$ as the attractor vector for the vertex v_i. Vertices belonging to the same cluster have attractor vectors that are close to each other. Any unsupervised clustering algorithm, such as k-means or single linkage clustering method, can be applied to find the desired number of (k) clusters. Because of the computational complexity of these clustering algorithms, we design a fast merging algorithm that can efficiently cluster vertices according to their attractor vectors.

Each element $x_j^{(*,i)}$ in $x^{(*,i)}$ is the probability value of the stationary state that the walking agent hits the vertex v_j when the seed vertex is v_i. The attractor vector $x^{(*,i)}$ is determined by the graph structure of the cluster that the initial vertex v_i has. Thus, vertices in the same cluster should have very similar attractor vectors. We first find the vertex that has the largest value in the vector $x^{(*,i)}$. Suppose $m = \arg\max_j \left(x_j^{(*,i)}\right)$, we call v_m the attractor vertex of vertex v_i. Grouping vertices by their attractor vertex can be done in a fast way (complexity of $O(1)$) using a dictionary data structure. After the grouping, each vertex is assigned to a cluster that is identified by the attractor vertex. However, it is possible that some vertices in one cluster do not have the same attractor vertex. This may happen when the cluster is large and the edge density in the cluster is low. We then apply the following cluster merging algorithm to handle this overclustering problem.

The vertices that have large values, which are determined by a threshold relative to $x_m^{(*,i)}$, in $x^{(*,i)}$ are called significant vertices for vertex v_i. If two vertices have large enough overlaps of their significant vertices, they should be grouped into the same cluster. From this observation, we first collect significant vertices for the found clusters. Then we merge clusters if their significant vertices overlap more than a half. Note that the attractor vertex and the significant vertices are always in the same cluster as the seed vertex. This is very useful when we use the multi-stage graph strategy.

Algorithm 2 shows the details of the merging phase of the LRW algorithm. Note that, for small graph data, we set the seed vertex set $\mathrm{Q} = \mathrm{V}$ and the initial clustering dictionary $\mathcal{D}$ to be empty.

Algorithm 2 LRW cluster merging phase

```
given feature vectors $x^{(*,i)}$ of the seed vertex set $Q$, where $i \in Q$, threshold $\tau$ such that $0 < \tau < 1$ ,
and initial clustering dictionary $\mathcal{D}$
for each feature vector $x^{(*,i)}$ where $i \in Q$
  1   find $m = \arg\max_j (x_j^{(*,i)})$ and add $i$ to an empty vertex set $\mathcal{S}$
  2   find all $j$ such that $x_j^{(*,i)} > \tau \cdot x_m^{(*,i)}$ and add them to an empty vertex set $\mathcal{F}$
  3   if $\mathcal{D}$ does not contain the key $m$
        add the pair $\langle \mathcal{S}, \mathcal{F} \rangle$ as the value associated with the key $m$ to dictionary $\mathcal{D}$
      else
        find the value $\langle \mathcal{S}_m, \mathcal{F}_m \rangle$ that is associated with the key $m$
        add $i$ to set $\mathcal{S}_m$ and merge $\mathcal{F}$ to $\mathcal{F}_m$ by $\mathcal{F}_m = \mathcal{F}_m \cup \mathcal{F}$
        update the value $\langle \mathcal{S}_m, \mathcal{F}_m \rangle$ of the key $m$ in the dictionary $\mathcal{D}$.
      end
for each pair of keys $m_1$ and $m_2$ in the dictionary $\mathcal{D}$
  1   get the value pairs of $\langle \mathcal{S}_{m_1}, \mathcal{F}_{m_1} \rangle$ and $\langle \mathcal{S}_{m_2}, \mathcal{F}_{m_2} \rangle$ that are associated with the key $m_1$ and
      $m_2$
  2   get the union of the two vertex sets $\mathcal{U} = \mathcal{F}_{m_1} \cap \mathcal{F}_{m_2}$
  3   if $|\mathcal{U}| > \frac{1}{2} \min (|\mathcal{F}_{m_1}|, |\mathcal{F}_{m_2}|)$
        (a) merge $\mathcal{S}_{m_2}$ to $\mathcal{S}_{m_1}$ , $\mathcal{S}_{m_1} = \mathcal{S}_{m_1} \cup \mathcal{S}_{m_2}$
        (b) merge $\mathcal{F}_{m_2}$ to $\mathcal{F}_{m_1}$ , $\mathcal{F}_{m_1} = \mathcal{F}_{m_1} \cup \mathcal{F}_{m_2}$
        (c) update $\langle \mathcal{S}_{m_1}, \mathcal{F}_{m_1} \rangle$ to the key $m_1$
        (d) delete the key $m_2$ and its associated value
for each key $m$ in $\mathcal{D}$, output $\mathcal{S}_m$ as the clustering result
```

Multi-stage strategy

For small graph data, we can do the LRW procedure on every vertex of the graph. So the seed vertex set $Q = V$. The graph clustering is completed after a graph exploring phase and a cluster merging phase. However, when the graph data is large, it is time-consuming to perform the LRW procedure from every vertex of the graph. A multi-stage strategy can be used to greatly reduce the number of required walkings. First, we start the LRW procedure from a randomly selected vertex set. After the first round of the graph exploring, some clusters can be found after the cluster merging phase. Next we generate a new seed vertex set by randomly selecting vertices from those vertices that have not been clustered. Then we do the graph exploration from the new seed vertex set. We repeat this procedure until all vertices are clustered.

Algorithm 3 shows the global graph clustering algorithm using the multi-stage strategy.

Algorithm 3 Global graph clustering algorithm using multi-stage strategy

```
given adjacency matrix $A$ of the graph $G(V, E)$, the exponent $r$ for the inflation function, threshold
value $\tau$, maximum number of iterations $T_{max}$ and a small enough value $\epsilon$
initialize the transition matrix $P$ by Eq. 4, vertex set $B = V$ and clustering dictionary $\mathcal{D} = \emptyset$
while B is not empty
    1  generate a vertex set $Q \subset B$ by randomly select vertices from $B$
    2  do the graph exploring from the vertex set $Q$ using algorithm 1 to get feature vectors $x^{(*,i)}$ for
       $i \in Q$
    3  given $x^{(*,i)}$ and $\mathcal{D}$, do the graph cluster merging using algorithm 2 to update the clustering
       dictionary $\mathcal{D}$
    4  for each key $m$ in $\mathcal{D}$, merge the attractor vertex set $\mathcal{F}_m$ to the cluster vertex set $\mathcal{S}_m$, $\mathcal{S}_m =$
       $\mathcal{S}_m \cup \mathcal{F}_m$
    5  get clustered vertex set $R = \bigcup_{m \in \mathcal{D}} \mathcal{S}_m$
    6  let $B = B \backslash R$
for each key $m$ in $\mathcal{D}$, output $\mathcal{S}_m$as the clustering result
```

LRW for local graph clustering problems

For the local graph clustering problems, the LRW procedure can efficiently find the cluster from a given seed vertex. To achieve this, we first perform graph exploring from the seed vertex in the same way as described in "Graph exploring phase" section. Let $x^{(*)}$ be the probability vector after the graph exploration. If a probability value in $x^{(*)}$ is large enough, the corresponding vertex is assigned to the local cluster without further computation. Similar to the global graph clustering algorithm, we use a relative threshold η that is related to the maximum value in $x^{(*)}$. Vertices whose probability values are greater than $\eta \cdot \max\left(x_j^{(*)}\right)$ are called significant vertices. The significant vertices are assigned to the local cluster directly. A small value of η will reduce the computational complexity, but may decrease the accuracy of the algorithm. Suitable values of η were experimentally found to be between 0.3 and 0.5.

The vertices with low probability values can either be outside the cluster or inside the cluster but with relatively low significance. Unlike [9, 15, 16], which involve a sweep operation and a cluster fitness function, we do another round of graph exploring from these insignificant vertices. After the second graph exploring is completed, we apply the cluster merging algorithm described in "Cluster merging phase" section.

Algorithm 4 presents the LRW local clustering algorithm.

Algorithm 4 LRW local graph clustering algorithm

```
given graph $G(V,E)$, seed vertex $v$, threshold values $\eta$ and $\tau$, where $0 < \eta < 1$ and $0 < \tau < 1$, the exponent $r$ of the inflation function, and maximum number of iterations $T_{max}$
1  do graph exploration starting from the seed vertex $v$ as described in algorithm 1 and get the attractor vector $x^{(*,v)}$
2  find vertices in $x^{(*,v)}$ for which $x_i^{(*,v)} > 0$ and collect the vertices in to set $S$
3  find $m = \arg\max_j(x_j^{(*,v)})$
4  split the set $S$ into two sets $S_1$ and $S_2$ such that $S_1 = \left\{j|x_j^{(*,v)} \geq \eta x_m^{(*,v)}\right\}$ and $S_2 = \left\{j|x_j^{(*,v)} < \eta x_m^{(*,v)}\right\}$
5  for each vertex in $S_2$ do graph exploration to get feature vectors $x^{(*,i)}$, where $i \in S_2$
6  do clustering merging according to the feature vectors of the vertex $v$ and the vertices in $S_2$ as described in algorithm 2 and find the cluster $S_3$ that contains the seed vertex $v$
7  return $S_1 \cup S_3$
```

Computational complexity

We first analyze the computational complexity of the LRW algorithm for the global graph clustering problem. We assume the graph $G(V,E)$ has clusters. Let $\bar{n}_c$ be the average cluster size—the number of vertices in the cluster, and C is the number of clusters. We have $\bar{n}_c \cdot C = n$. Note $C \ll n$. The most time-consuming part of the algorithm is the graph exploring phase. For each vertex, every iteration involves a multiplication of the transition matrix P and the probability vector x. The LRW procedure visits not only the vertices in the cluster but also a certain amount of vertices close to the cluster. Let γ be the coefficient that indicates how far the LRW procedure explores the graph before it converges. Notice the maximum number of nonzero elements in a probability vector is $1/\epsilon$. Let J denote the number of vertices that the LRW procedure visits in each iteration, thus $J = \min(\gamma\bar{n}_c, 1/\epsilon)$. Thus the transition step at each iteration has complexity of $O(J\bar{n}_c)$. The inflation and normalization steps, which operate on the probability vector x, have the complexity of $O(J)$. Let K be the number of iterations for the LRW procedure to converge. So, the computational complexity for a complete LRW procedure on each vertex is $O(KJ\bar{n}_c)$. For a global clustering problem when performing the LRW procedure on every vertex, the graph exploration phase has a complexity of $O(KJ\bar{n}_c n)$. In the worst case, the algorithm has a complexity of $O(n^3)$. This is an extremely rare case and it only happens when the graph is small; does not have a cluster structure; and the edge density is high. This worst case scenario is identical to the MCL algorithm [12]. Notice that the variables J and K have upper bounds and $\bar{n}_c$ is determined by the graph structure, the algorithm has a complexity of $O(n)$ for big graph data.

The computational complexity of the cluster merging phase involves merging clusters that were found using the attractor vertices. This merging requires $\binom{C}{2}$ times of set comparison operations, where C is the number of clusters found by the attractor vertices. The complexity of this phase is roughly $O(C^2)$. This does not impose a significant impact to the overall complexity of the algorithm, since $C \ll n$. The time spent in this phase is often negligible. Experiments show that the clusters found using the attractor vertices are close to the final results. For applications where speed is more important than accuracy, the cluster merging phase can be left out.

When the LRW algorithm is used in local graph clustering problems, the first graph exploration (started from the seed vertex) has a complexity of $O(KJ\bar{n}_c)$. After the first graph exploration, there are LJ vertices to be further explored, where L is related to the threshold η and $L < 1$. The overall complexity of the LRW local clustering algorithm is thus $O(LKJ^2\bar{n}_c)$.

The LRW algorithm is a typical example of embarrassingly parallel paradigm. In the graph exploring phase, each random walk can be executed independently. Therefore it can be entirely implemented in a parallel computing environment such as a high-performance computing system. The time spent for graph exploring phase decreases roughly linearly with respect to the number of available computing resources. The two-phase design also fits the MapReduce programming model and can easily be adapted into any MapReduce framework [23].

Experiments

The LRW algorithm uses the following parameters: inflation exponent r, maximum number of iterations T_{max}, small value ϵ, merging threshold τ and local clustering threshold η. In practice, except the inflation exponent r, the values of the other parameters have little impact to the final results. The inflation power r should be chosen according to the density of the graph. A sparse graph should use a smaller value of r, though $r = 2$ is suitable for most real world graphs. In our experiments, we chose $r = 2$ unless otherwise specified. The other parameters have been set as: $T_{max} = 100, \epsilon = 10^{-5}$ and $\tau = \eta = 0.3$. We will show the impact of some parameters in "The sensitivity analysis of the parameters" section.

Simulated data for global graph clustering problem

We first show the performance of the LRW algorithm using simulated graph data. The simulated graph is generated using the Erdos-Renyi model [24] with some modifications to generate clusters. Using the ground truth of the cluster structure, we can evaluate the performance of graph clustering algorithms. This kind of simulated data are widely used in the literature [5, 25–27].

The graphs are generated by the model $G(n, p, c, q)$ where c is the number of clusters, n is the number of vertices, p is the probability of the link between two vertices, and $q = d_{in}/d_{out}$ is the parameter that indicates the strength of the cluster structure, where d_{in} is the expected number of edges linking one vertex to other vertices inside the same cluster, and d_{out} is the expected number of edges linking a given vertex to other vertices in other clusters. Larger q indicates stronger cluster structure. When $q = 1$, each vertex has equal probability that it links to vertices that are inside and outside the cluster—the graph has a very weak cluster structure. Let d be the expected of degree of a vertex. So, $d = d_{in} + d_{out} = p(n - 1)$. We use this model to generate graphs that consist of c clusters and each cluster has the same number of vertices. For each pair of vertices, we link them with the probability $\frac{qpc(n-1)}{(q+1)(n-c)}$ if they belong to the same cluster, and the probability of $\frac{pc(n-1)}{n(q+1)(c-1)}$ if they belong to different clusters.

We use the normalized mutual information (NMI) to evaluate the clustering result against the ground truth [28, 29]. We first calculate the confusion matrix where each row is a cluster found by the clustering algorithm and each column is a cluster in the ground truth. The entries in the confusion matrix are the cardinality of the intersect set of the row cluster and the column cluster. Let N_{ij} be the values at the i-th row and the j-th column, N_{i-} the sum of the values at the i-th row, N_{-j} the sum of the values at the

j-th column, N the total number of vertices, C_A the number of clusters that the clustering algorithm found (number of rows), and C_G the number of clusters in the ground truth (number of columns). The NMI is calculated as follows:

$$NMI = \frac{-2\sum_{i=1}^{C_A}\sum_{j=1}^{C_G} N_{ij}\log\left(N_{ij}N/N_{i-}N_{-j}\right)}{\sum_{i=1}^{C_A} N_{i-}\log\left(N_{i-}/N\right)+\sum_{j=1}^{C_G} N_{-j}\log\left(N_{-j}/N\right)}, \tag{11}$$

where $0 \leq NMI \leq 1$. If the clustering algorithm returns the exact same cluster structure as the ground truth, $NMI = 1$. Notice, NMI is not a symmetric evaluation metric. If an algorithm assigns all vertices into one cluster ($C_A = 1$), then NMI value is 0. On the other hand, if an algorithm assigns each vertex to its own cluster ($C_A = N$), then $NMI > 0$.

We generated graphs by choosing $n = 128$ and $d = 16$. The number of the generated clusters is 4 and each cluster contains 32 vertices. We varied the ratio q and evaluated the performance of the LRW algorithm against Girvan-Newman (GN) [27], Louvain [6], Infomap [30] and MCL [12] algorithms. GN clusters a graph by iteratively removing edges according to their "betweenness" measures. Louvain optimizes the modularity measure of a graph using a greedy search paradigm. Infomap is another modularity-based algorithm. It starts with each vertex in its own cluster and iteratively merges the clusters, moves vertices between clusters or splitting a cluster until no better modularity measure can be found. MCL is a random walk based algorithm that also involves inflation operation. The differences between the MCL algorithm and the proposed one are explained in "Definitions" section.

Two simulated graphs are shown in Fig. 1, where the clusters are colored differently and the graphs are visualized by force-directed algorithms.

The comparative results are given in Table 1, where the number of clusters found by the algorithms is placed between parentheses.

From the results, Louvain is the best performing algorithm and the LRW algorithm comes as the second. It can be seen that the LRW algorithm can find the correct structure if the graph has a strong cluster structure. When the cluster structure diminishes as q decreases, the walking agents quickly spread to the whole graph before the contraction dominates. Thus, the LRW algorithm returns the whole graph as one cluster. This behavior is beneficial when we need to find the true clusters in a big graph. The GN and Louvain algorithms are more like graph partition algorithms. They optimize certain cluster fitness functions using the whole graph data. They tend to partition the graph into clusters even though the cluster structure is weak. That explains their better NMI scores in Table 1 when q is small.

We use real graph data to evaluate the performance of the LRW global clustering algorithm on heterogeneous graphs. Details of the experiments and the results are given in "Real world data" section.

Simulated data for local graph clustering problems

In this section, we compare the LRW algorithm with other local clustering algorithms. The test graphs are generated using the protocol defined in [26]. To simulate the data that are close to real world graphs, the vertex degree and the cluster size are chosen to follow the power law. Each test graph contains 2048 vertices. The vertex degree has the

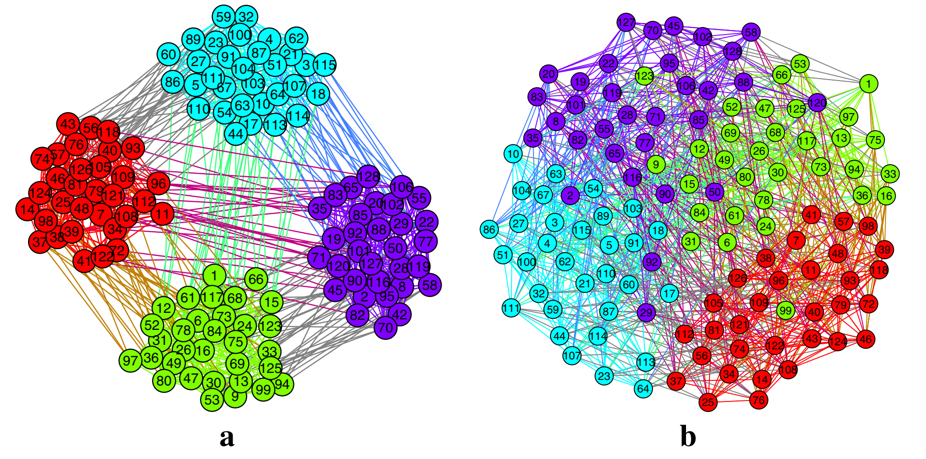

Fig. 1 Simulated graphs. **a** $q = 4$, **b** $q = 1.22$. The clusters are colored differently and the *graphs* are visualized by a force-directed method

Table 1 The NMI values and the numbers of clusters of the clustering results on simulated graph data

q	GN	Louvain	Infomap	MCL	LRW
4.0	0.975 (4)	1.0 (4)	1 (4)	1 (4)	1.0 (4)
3.0	1 (4)	1.0 (4)	1 (4)	1 (4)	1.0 (4)
2.33	0.950 (4)	1.0 (4)	1 (4)	0.860 (7)	1.0 (4)
1.86	0.900 (4)	1.0 (4)	1 (4)	0.478 (95)	1.0 (4)
1.5	0.890 (4)	1.0 (4)	0 (1)	0.453 (119)	0.975 (4)
1.22	0.593 (4)	0.771 (5)	0 (1)	0.444 (128)	0 (1)
1	0.232 (4)	0.304 (7)	0 (1)	0.444 (128)	0 (1)

minimum value of 16 and the maximum value of 128. The minimum and maximum cluster sizes are 16 and 256, respectively. Similar to the previous section, the inbound-outbound ratio q defines the strength of the cluster structure.

The competing algorithms are criteria-based algorithms that optimize a fitness function using either the greedy search or the simulated annealing optimization method. Let vertex set K be the cluster that contains the seed vertex. $K^c = V \backslash K$ is the complement vertex set of K. Let function $a(\cdot)$ be the total degree of a vertex set, that is

$$a(S) = \sum_{i \in S, j \in V} A_{ij}, \tag{12}$$

where A_{ij} are the entries of the adjacency matrix. The cut of the cluster K is defined as

$$c(K) = \sum_{i \in K, j \in K^c} A_{ij}. \tag{13}$$

The following are the definitions of the fitness functions.

Cheeger constant (conductance):

$$f(K) = \frac{c(K)}{\min\left(a(K), a(K^c)\right)} \tag{14}$$

Normalized cut:

$$f(K) = \frac{c(K)}{a(K)} + \frac{c(K)}{a(K^c)} \tag{15}$$

Inverse relative density:

$$f(K) = \frac{|E| - a(K) + c(K)}{a(K) - c(K)} \tag{16}$$

Different local clustering algorithms are used to find the cluster that contains the seed vertex. The Jaccard index is used to evaluate the performance of each algorithm. Let K be the set of vertices that an algorithm finds and $\mathcal{T}$ be the ground truth cluster that contains the seed vertex. The Jaccard index is defined as

$$J = \frac{|K \cap \mathcal{T}|}{|K \cup \mathcal{T}|} \tag{17}$$

We generated 10 test graphs for each inbound-outbound ratio q. From each generated graph, we randomly picked 20 vertices as seeds. For each algorithm and each inbound-outbound ratio q, we computed the Jaccard index for each seed and took the average of the 200 Jaccard indices. The results are shown in Table 2, where “Che” stands for the Cheeger constant (conductance) fitness function; “NCut” stands for the normalized cut fitness function; “IRD” stands for the inverse relative density; the ending letter “G” stands for the greedy search method; and the ending letter “S” stands for the simulated annealing method.

From the results, it is obvious that the LRW algorithm clearly outperforms other methods when the graph has a clear cluster structure. For the same reason explained in "Simulated data for global graph clustering problem" section, it does not give good result if the cluster structure is weak. This is the main difference between the LRW algorithm and graph partition algorithms.

Real world data

In this section, we evaluate the performance of the LRW algorithm on some real world graph data.

Zachary’s karate club

We first do clustering analysis on the Zachary’s karate club graph data [31]. This graph is a social network of friendship in a karate club in 1970. Each vertex represents a club member and each edge represents the social interaction between the two members. During the study, the club split into two smaller ones due to the conflicts between the administrator and the coach. The graph data have been regularly used to evaluate the performance of the graph clustering algorithms [27, 30, 32]. The graph contains 34 vertices and 78 edges. We applied the LRW algorithm on this graph and the result shows two clusters that are naturally formed. Figure 2 shows the clustering result, where clusters are illustrated using different colors.

Table 2 Jaccard index of local graph clustering results on the simulated graphs

q	CheG	CheS	NCutG	NCutS	IRDG	IRDS	LRW
4.0	0.753	0.840	0.752	0.820	0.753	0.830	*0.945*
3.0	0.671	0.812	0.671	0.801	0.671	0.798	*0.927*
2.33	0.668	0.774	0.668	0.758	0.668	0.776	*0.880*
1.86	0.593	0.650	0.593	0.681	0.593	0.684	*0.823*
1.5	0.492	0.630	0.493	0.629	0.492	0.609	*0.660*
1.22	0.444	0.544	0.437	*0.549*	0.444	0.529	0.504
1	0.298	0.410	0.296	*0.452*	0.298	0.424	0.295

As the figure shows, the LRW algorithm finds the two clusters of the Zachary's karate club. Actually the two clusters perfectly match the ground truth—how the club was split in 1970.

Clustering results of the GN, Louvain, Infomap and MCL algorithms are given in Additional file 1.

Ego-Facebook graph data

The second data we used is the ego-Facebook graph data [33]. The social network website Facebook allows users to organize their friends into "circles" or "friend lists" (for example, friends who share common interests). This data was collected from volunteer Facebook users for researchers to develop automatic circle finding algorithms. Ego-network is the network of an end user's friends. The ego-Facebook graph is a combination of ego-networks from 10 volunteer Facebook users. There are 4039 vertices and 88234 edges in the graph.

We applied the LRW, GN [27], Louvain [6], Infomap [30] and MCL [12] graph clustering algorithms to this data. To compare the results, we generated the ground truth clustering by combining the vertices in the "circles" of each volunteer user. So, the ground truth contains 10 clusters. If a vertex appears in the circles of more than one volunteer, we assign the vertex to all of these ground truth clusters. We evaluated the number of clusters and the NMI values of the results that each competing algorithm generated.

We also calculated the mean conductance (MC) value of the clustering results. The conductance value of a cluster is calculated using Eq. 14. We then took the mean of all the conductance values of the clusters that an algorithm finds. Smaller MC values indicate better clustering results. Note that MC tends to favor smaller numbers of clusters in general. If the numbers of clusters are roughly the same, MC values give good evaluation of the clustering results. It is also worth noting that MC value is capable of evaluating clustering algorithms without the ground truth. We shall use this metric in later experiments where the ground truth is not available.

The MC scores, NMI scores and the number of clusters found by each algorithm are reported in Table 3. A italic font indicates the best score among all competing algorithms.

The results show that the random-walk-based algorithms—LRW and MCL—are able to find the correct cluster structure of the data. Other criteria-based algorithms are sensitive to trivial disparities of the graph structure and are likely to overcluster the data.

The clustering result of the LRW algorithms is shown in Fig. 3.

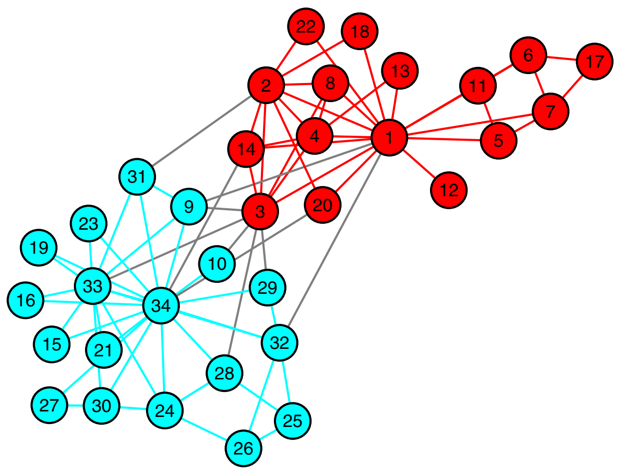

Fig. 2 Clustering result of the karate club graph data. The two clusters found by the LRW algorithm match the ground truth—how the club was split in 1970

Table 3 Global graph clustering results on the ego-Facebook graph

	GN	Louvain	Infomap	MCL	LRW
Mean conductance	0.156	0.133	0.397	0.0882	*0.0770*
NMI	0.778	0.796	0.723	0.908	*0.910*
Number of clusters	16	19	76	*10*	*10*

Clustering results of other algorithms are shown in the Additional file 1.

Heterogeneous graph data

To evaluate the performance of the LRW algorithm on real heterogeneous graph data, we selected 5 graph data from the collection of the KONECT project [34]. The graph data are selected from different categories and the size of the graph data varies from small to medium. The properties and the references of the test graphs are shown in Table 4.

Since there is no ground truth available for these test data, we evaluated each clustering algorithm by the mean conductance (MC) values. The results are in Table 5. The best MC scores are shown in a italic font. The numbers of clusters found by each algorithm are placed between parentheses. We also plot the clustered graphs in which the vertices are located using a force-directed algorithm and colored according to their associated clusters. These clustered graphs are given in the Additional file 1 for subjective evaluation.

The reactome and the infectious graphs have low density. We chose the inflation exponent $r = 1.2$ to prevent overclustering the data. For other graph data, the default value $r = 2$ is used.

Based on the MC scores and the visualized clustering results, the LRW algorithm achieves a superior clustering performance in most of the cases. Note that the reactome data has a weak cluster structure, thus the LRW algorithm has difficulty to find a good partition for it.

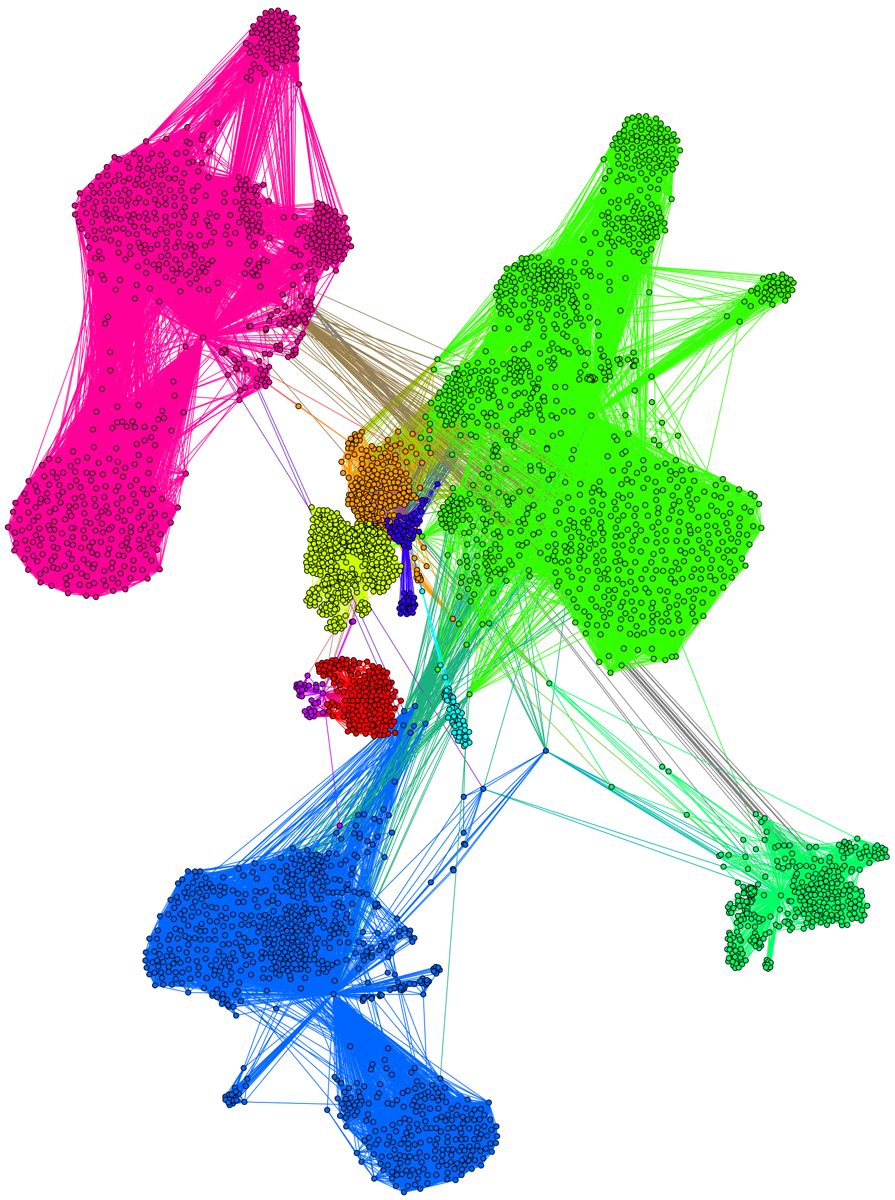

Fig. 3 Clustering result on the ego-Facebook graph data. *Different colors* indicate the clusters found by the LRW algorithm and the *graph* is visualized by a force-directed method

The sensitivity analysis of the parameters

The proposed LRW algorithm depends on a number of parameters to perform global and local graph clustering. In this section, we perform the sensitivity analysis on the parameters.

We use both simulated and real-world graphs in our experiments. Test graph G1, G2 and G3 are similar to those used in "Simulated data for global graph clustering problem" section except that we vary the density of each graph. The expected degree d, which is a measure of the graph density, of graph G1, G2 and G3 are 12, 16 and 20 respectively and q is set to be 1.86 for all test graphs. Test graph G4 is generated in the same way as described in "Simulated data for local graph clustering problems" section. The

Table 4 Properties of the heterogeneous graphs used for testing

	Vertices	Edges	Category	Reference
Dolphins	62	156	Animal	[35]
Arenes-jazz	198	2742	Human social	[36]
Infectious	410	2765	Human contact	[37]
Polblogs	1490	19,090	Hyperlink	[38]
Reactome	6229	146,160	Metabolic	[39]

Table 5 Global graph clustering results on the real heterogeneous graph data

	GN	Louvain	Infomap	MCL	LRW
Dolphins	0.425 (4)	0.440 (5)	0.487 (6)	0.675 (12)	*0.347* (4)
Arenes-jazz	0.485 (4)	0.455 (4)	0.577 (7)	0.529 (5)	*0.364* (4)
Infectious	*0.162* (5)	0.214 (6)	0.465 (17)	0.673 (40)	0.175 (5)
Polblogs	0.524 (12)	0.501 (11)	0.727 (36)	0.777 (45)	*0.427* (11)
Reactome	0.108 (110)	*0.099* (114)	0.315 (248)	0.478 (352)	0.221 (191)

ego-Facebook graph data in "Ego-Facebook graph data" section is used as an example of real-world graphs. We performed global graph clustering on these test graphs using the LRW algorithm with different parameters. the NMI scores are used to evaluate the performance of the algorithm. The experiments using simulated graph data were repeated 10 times and the average NMI scores and the average number of clusters are reported.

As described in "Limited random walk on general graphs" section, the most important parameter of the LRW algorithm is the inflation exponent r. We first set $T_{max} = 100$, $\epsilon = 10^{-5}$, $\tau = 0.3$ and vary the inflation exponent r. Table 6 shows the NMI scores and the number of clusters reported by the LRW algorithm with different values of r.

The test results show the relationship among the inflation exponent r, the density of the test graphs and the performance the LRW algorithm. A large inflation exponent r may overcluster the data as the results on graph G1 and G2 shows. It can be easily noticed that the LRW algorithm is not sensitive to the choice of r for test graph G4 and the ego-Facebook graph. These graphs, and almost all real-world graphs, are more heterogeneous than the simulated graphs G1, G2 and G3. The LRW algorithm performs better on this type of graph since the attractor vertices and significant vertices are more stable on these graphs.

The parameter T_{max} sets a limit on the number of iterations for the LRW procedure to converge. According to our experiments, value 100 is large enough to ensure the convergence of almost all cases. For example, only 3 out of 88,234 LRW procedures do not converge within 100 iterations on the ego-Facebook graph. A few exceptional cases has no impact on the final clustering results. The parameter ϵ is used to remove small values in the probability vector thus decrease the computational complexity. It has no impact on the final clustering result as long as the value is small enough, for example $\epsilon < 10^{-4}$.

We also conducted the experiments by varying the threshold value τ from 0.1 to 0.5. The NMI scores and the number of clusters found by the LRW algorithm with different τ values are almost identical to the values in the corresponding cells in Table 6. This indicates that the choice of τ has very little impact on the clustering performance.

Table 6 The NMI scores and the number of clusters by the different inflation exponent values

r	G1	G2	G3	G4	Facebook
1.2	0 (1)	0 (1)	0 (1)	0.155 (5)	0.902 (10)
1.4	0 (1)	0 (1)	0 (1)	0.927 (22.7)	0.906 (10)
1.6	0 (1)	0 (1)	0 (1)	1.0 (23)	0.908 (11)
1.8	1 (4)	1 (4)	1 (4)	1.0 (20.8)	0.910 (10)
2	0.971 (4.6)	1 (4)	1 (4)	1.0 (25)	0.910 (10)
2.4	0.868 (7.0)	0.990 (4.2)	1 (4)	1.0 (21.8)	0.910 (10)
3	0.822 (6.3)	0.962 (4.8)	0.988 (4.2)	1.0 (24.3)	0.910 (10)

According to these results, one only needs to choose a proper inflation exponent r to use the LRW algorithms. Other parameters can be chosen freely from a wide range of reasonable values. $r = 2$ is suitable for most of graphs and is preferable because of the computational advantage.

Big graph data

In this section, we apply the LRW algorithm on real-world big graph data and show the computational advantage of its parallel implementation. The test graphs were received from the SNAP graph data collection [40, 41]. These graphs are from major social network services and E-commerce companies. We use the high quality communities that either created by users or the system as ground truth clusters. The details of the high quality communities are described in [41]. The Rand index is used to evaluate the results of the proposed clustering algorithm. To generate positive samples, we randomly picked 1000 pairs of vertices, where the vertices in each pair come from the same cluster in the ground truth. Negative samples consist of 1000 pairs of vertices, where the vertices from each pair come from different clusters in the ground truth. The Rand index is defined as

$$\mathrm{RI} = \frac{TP + TN}{N}, \tag{18}$$

where TP is the number of true positive samples, TN is the number of true negative samples, and N is the total number of samples.

Since none of the competing algorithms used in previous sections can complete this task due to the large size of the data, we only report the results from the LRW algorithm. Table 7 shows the size of the test graphs, the time spent on the graph exploration phase, the number of CPU cores and the amount of memory used for graph exploration, the time spent on cluster merging phase, the number of clusters that the LRW algorithm finds and the Rand index of the clustering results. In this experiment, multiple CPU cores were used for graph exploration and one CPU core was used for clustering merging.

Table 7 shows that the LRW algorithm is able to find clusters from large graph data with a reasonable computing time and memory usage. The Rand index values indicate that the clusters returned by the LRW algorithm match well the ground truth. The time spent on the graph exploration phase is inversely proportional to the number of CPU

Table 7 Clustering performance of real-world big graph data

	com-Amazon	com-Youtube	com-LiveJournal	com-Orkut
Vertices	334,863	1,134,890	3,997,962	3,072,441
Edges	925,872	2,987,624	34,681,189	117,185,083
CPU cores (graph exploration)	12	96	96	96
Memory per CPU core	4G	4G	8G	8G
Graph exploration (in hours)	0.83	3.08	17.4	24.0
Cluster merging (in hours)	0.20	1.34	9.44	2.53
Clusters	37,473	170,569	381,246	165,624
Rand index	0.908	0.755	0.951	0.751

cores. Computational time can be further reduced if more computing resources are available. The proposed algorithm can efficiently handle graphs with millions of vertices and hundreds of millions of edges. For even larger graphs that exceed the memory limit for each computing process, a mechanism that retrieves part of the graph from a central storage can be used. Since the LRW procedure is capable of exploring a limited number of vertices that are near a seed vertex, the algorithm can cluster much larger graphs if such a mechanism is implemented.

Conclusions

In this paper, we proposed a novel random-walk-based graph clustering algorithm, the so-called LRW. We studied the behavior of the LRW procedure and developed the LRW algorithms for both global and local graph clustering problems. The proposed algorithm is fundamentally different from previous random-walk-based algorithms. We use the LRW procedure to find attracting vertices and use them as features to cluster vertices in a graph. The performance of the LRW algorithm was evaluated using simulated graphs and real-world big graph data. According to the results, the proposed algorithm is superior to other well-known methods.

The LRW algorithm can be efficiently used in both global and local graph clustering problems. It finds clusters from a big graph data by only locally exploring the graph. This is important for extreme large data that may not even fit in a single computer memory. The algorithm contains two phases—the graph exploring phase and the cluster merging phase. The graph exploring phase is the most critical part and also the most time-consuming part of the algorithm. This phase can be implemented in embarrassingly parallel paradigm. The algorithm can easily be adapted to any MapReduce framework.

From our experiments, we also noticed the limitations of the LRW algorithm. First, when used as a global clustering algorithm, the computational complexity can be high, especially when the graph cluster structure is weak. This is due to the fact that the graph may be analyzed multiple times during the graph exploration phase, if we perform the LRW procedure from every vertex of the graph. However, using the multi-stage strategy can dramatically reduce the computation time. Second, if the cluster structure is weak, the LRW algorithm may return the whole graph as one cluster—though this behavior is desired in many cases.

The experiments show that the performance of the proposed LRW graph clustering algorithm is not sensitive to any parameter except the inflation exponent r, especially when the graph is not heterogeneous. For future research, we will further improve the LRW algorithm so that it can optimally select the inflation function that best suits the problem at hand.

Additional file

Additional file 1. Clustering results on graphs used in the experiments of various methods.

Abbreviations

GN: Girvan-Newman; LRW: limited random walk; MCL: Markov clustering algorithm; MC: mean conductance; NMI: normalized mutual information.

Authors' contributions

HZ carried out the conception and design of the study, participated in the analysis and interpretation of data, and was involved in drafting and revising the manuscript. JR, SK and MG made substantial contributions to the design of the study, the analysis and interpretation of the data, and were involved in critically reviewing the manuscript. All authors read and approved the final manuscript.

Author details

[1] Department of Signal Processing, Tampere University of Technology, Korkeakoulunkatu 1, 33101 Tampere, Finland.
[2] Electrical Engineering Department, College of Engineering, Qatar University, 2713, Al Hala St, Doha, Qatar.

Competing interests

The authors declare that they have no competing interests.

Received: 23 September 2016 Accepted: 15 November 2016
Published online: 01 December 2016

References

1. Benson AR, Gleich DF, Leskovec J. Higher-order organization of complex networks. Science. 2016;353(6295):163–6.
2. Schaeffer SE. Graph clustering. Comput Sci Rev. 2007;1(1):27–64.
3. Lambiotte R, Delvenne JC, Barahona M. Random walks, Markov processes and the multiscale modular organization of complex networks. IEEE Trans Netw Sci Eng. 2014;1(2):76–90.
4. He P, Xu X, Hu K, Chen L. Semi-supervised clustering via multi-level random walk. Pattern Recognit. 2014;47(2):820–32.
5. Newman ME. Fast algorithm for detecting community structure in networks. Phys Rev E. 2004;69(6):066133.
6. Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech Theory Exp. 2008;2008(10):10008.
7. Clauset A, Newman ME, Moore C. Finding community structure in very large networks. Phys Rev E. 2004;70(6):066111.
8. Waltman L, van Eck NJ. A smart local moving algorithm for large-scale modularity-based community detection. Eur Phys J B. 2013;86(11):1–14.
9. Spielman DA, Teng SH. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. arXiv:0809.3232; 2008.
10. Qiu H, Hancock ER. Graph matching and clustering using spectral partitions. Pattern Recognit. 2006;39(1):22–34.
11. Spielman D, Teng S. Spectral sparsification of graphs. SIAM J Comput. 2011;40(4):981–1025.
12. Dongen S. Graph clustering by flow simulation. PhD thesis, Universiteit Utrecht, Utrecht, The Netherlands; 2000.
13. Macropol K, Can T, Singh AK. RRW: repeated random walks on genome-scale protein networks for local cluster discovery. BMC Bioinform. 2009;10(1):283.
14. Xin Y, Xie Z-Q, Yang J. The adaptive dynamic community detection algorithm based on the non-homogeneous random walking. Phys A Stat Mech Appl. 2016;450:241–52.
15. Chung F, Kempton M. A local clustering algorithm for connection graphs. In: Algorithms and models for the web graph; 2013. p. 26–43.
16. Macko P, Margo D, Seltzer M. Local clustering in provenance graphs. In: Proceedings of the 22nd ACM international conference on conference on information & knowledge management; 2013. p. 835–840.
17. Andersen R, Chung F, Lang K. Using pagerank to locally partition a graph. Internet Math. 2007;4(1):35–64.
18. Buhler T, Rangapuram SS, Setzer S, Hein M. Constrained fractional set programs and their application in local clustering and community detection. arXiv:1306.3409; 2013.
19. Zhu ZA, Lattanzi S, Mirrokni V. A local algorithm for finding well-connected clusters. In: Proceedings of the 30th international conference on machine learning (ICML-13); 2013. p. 396–404.
20. Norris JR. Markov chains applied probability and stochastic networks; 1998)
21. Harel D, Koren Y. On clustering using random walks. In: Hariharan R, Vinay V, Mukund M, et al., editors. Theoretical computer science., Lecture notes in computer scienceBerlin: Springer; 2001. p. 18–41.

22. Cai B, Wang H, Zheng H, Wang H. An improved random walk based clustering algorithm for community detection in complex networks. In: 2011 IEEE international conference on systems, man, and cybernetics (SMC); 2011. p. 2162–2167.
23. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
24. Newman M. Networks: an introduction. 1st ed. New York: Oxford; 2010.
25. Danon L, Díaz-Guilera A, Arenas A. The effect of size heterogeneity on community identification in complex networks. J Stat Mech Theory Exp. 2006;2006(11):11010.
26. Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection algorithms. Phys Rev E. 2008;78(4):046110.
27. Newman ME, Girvan M. Finding and evaluating community structure in networks. Phys Rev E. 2004;69(2):026113.
28. Ana LNF, Jain AK. Robust data clustering. In: Proceedings 2003 IEEE computer society conference on computer vision and pattern recognition, vol. 2; 2003. p. 128–1332.
29. Danon L, Diaz-Guilera A, Duch J, Arenas A. Comparing community structure identification. J Stat Mech Theory Exp. 2005;2005(09):09008.
30. Rosvall M, Bergstrom CT. Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci. 2008;105(4):1118–23.
31. Zachary WW. An information flow model for conflict and fission in small groups. J Anthropol Res. 1977;1:452–73.
32. Sahai T, Speranzon A, Banaszuk A. Hearing the clusters of a graph: a distributed algorithm. Automatica. 2012;48(1):15–24.
33. Leskovec J, Mcauley JJ. Learning to discover social circles in ego networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in neural information processing systems 25; 2012. p. 539–547.
34. Kunegis J. Konect—the Koblenz network collection. In: Proceedings of international conference on World Wide Web companion; 2013. p. 1343–1350.
35. Lusseau D, Schneider K, Boisseau OJ, Haase P, Slooten E, Dawson SM. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. Behav Ecol Sociobiol. 2003;54(4):396–405.
36. Gleiser P, Danon L. Community structure in jazz. Adv Complex Syst. 2003;06(04):565–73.
37. Isella L, Stehlé J, Barrat A, Cattuto C, Pinton JE, van den Broeck W. What's in a crowd? Analysis of face-to-face behavioral networks. J Theor Biol. 2011;271(1):166–80.
38. Adamic LA, Glance N. The political blogosphere and the 2004 US election: divided they blog. Proceedings of the 3rd international workshop on link discovery., LinkKDD '05New York: NY, USA; 2005. p. 36–43.
39. Croft D, Mundo AF, Haw R, Milacic M, Weiser J, Wu G, Caudy M, Garapati P, Gillespie M, Kamdar MR, Jassal B, Jupe S, Matthews L, May B, Palatnik S, Rothfels K, Shamovsky V, Song H, Williams M, Birney E, Hermjakob H, Stein L, D'Eustachio P. The reactome pathway knowledgebase. Nucleic Acids Res. 2013;1102:472–7.
40. Leskovec J. Stanford large network dataset collection.
41. Yang J, Leskovec J. Defining and evaluating network communities based on ground-truth. arXiv:1205.6233; 2012.