Graph Clustering Using Early-Stopped Random Walks

Małgorzata Lucińska[1(✉)] and Sławomir T. Wierzchoń[2]

[1] Kielce University of Technology, Kielce, Poland
lucinska@tu.kielce.pl
[2] Institute of Computer Science Polish Academy of Sciences, Warsaw, Poland

Abstract. Very fast growth of empirical graphs demands clustering algorithms with nearly-linear time complexity. We propose a novel approach to clustering, based on random walks. The idea is to relax the standard spectral method and replace eigenvectors with vectors obtained by running early-stopped random walks. We abandoned iterating the random walk algorithm to convergence but instead stopped it after the time that is short compared with the mixing time. The computed vectors constitute a local approximation of the leading eigenvectors. The algorithm performance is competitive to the traditional spectral solutions in terms of computational complexity. We empirically evaluate the proposed approach against other exact and approximate methods. Experimental results show that the use of the early stop procedure does not influence the quality of the clustering on the tested real world data sets.

Keywords: Graph clustering · Random walks · Convergence rate

1 Introduction

Graph partitioning is the problem of dividing a graph into groups, with dense connections within groups and only sparser connections between them. It is an ubiquitous technique which has applications in many fields of computer science and engineering, such as image segmentation [16], in the World Wide Web [7], in biochemical neural networks [24], and in bioinformatics for protein family classification [5] among others.

Graph partitioning methods can also be categorized as either global or local optimizers. Global methods try to discover structure of the whole graph. In local partitioning, the goal is to find a group containing a given seed vertex. Hence, essentially, it is the task of finding a bipartition of the graph into two vertex sets.

A class of algorithms widely used to detect clusters in graphs is based on spectral methods, [12]. Here, to partition a graph, the eigenvectors of a suitably chosen matrix are used. This matrix represents connectivity between vertices

© IFIP International Federation for Information Processing 2016
Published by Springer International Publishing Switzerland 2016. All Rights Reserved
K. Saeed and W. Homenda (Eds.): CISIM 2016, LNCS 9842, pp. 416–428, 2016.
DOI: 10.1007/978-3-319-45378-1_37

to be grouped. The eigenvector associated to the second smallest eigenvalue of the matrix is called the Fiedler vector, named after Fiedler for his contributions to algebraic graph theory [6]. The Fiedler vector is one of the most important spectral characteristics, carrying information about structural properties of the graph.

As spectral algorithms represent global partitioning methods, they are relatively slow and thus have been mostly superseded by faster local algorithms, see e.g. [13]. Moreover, global solutions are infeasible in cases of graphs with only partly known structure, as for example the WWW network.

We propose a solution that combines the spectral method with lazy random walks. In our algorithm the Fiedler vector is approximated by running lazy random walk. After choosing a seed vertex A, for each vertex i, we update the probability of being absorbed by this A before the random walker will be captured by other distant vertex. The method is local in the sense that we obtained a rough, local approximation of the Fiedler vector by early stopping the updating process. Random walk process is not run till the convergence but stopped after the maximum probability change does not exceed an established threshold. Our method allows for bipartitioning graphs in times that scale almost linearly with their size. The procedure is motivated by machine learning practices, that try to find not an exact solution but more general one, not influenced by noise or mistakes. We have verified our approach experimentally, using real-world testing sets, including protein networks.

In Sect. 2 the notation and related terms are presented, later we give basic facts concerning random walks. The next section describes related works. Our solution is presented in details in Sect. 5. Then, in Sect. 6, we compare performance of our algorithm with another solution. Finally, in Sect. 7, the main conclusions are drawn.

2 Notation and Related Terms

Let $G = (V, E)$ be a graph with the vertex set V and the edge set $E \subseteq V^2$. The graph is undirected, i.e. each edge $e \in E$ is an unordered pair of nodes from V, and simple, i.e. the graph has no loops.

The matrix $\mathbf{S} = [s_{ij}]$ plays a role of the affinity matrix for G. The degree of a node i equals $d(i) = \sum_j s_{ij}$, and $\mathbf{D}$ is the diagonal matrix with $d(i)$'s on its diagonal. In our solution we use an adjacency matrix $\mathbf{A} = [a_{ij}]$ with $a_{ij} = 0$ if there is an edge between i-th and j-th vertex and $a_{ij} = 0$.

A clustering $\mathcal{C} = \{C_1, C_2, ...C_K\}$ is a partitioning of V into the nonempty mutually disjoint subsets $C_1, C_2, ...C_K$.

The Laplacian matrix associated with graph G is the $n \times n$ matrix $\mathbf{L} = \mathbf{D} - \mathbf{S}$. Since $\mathbf{S}$ is a symmetric matrix, $\mathbf{L}$ is also symmetric. The normalized Laplacian, is defined as: $\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. The right eigenvector associated to the second smallest eigenvalue of the Laplacian matrix is called the Fiedler vector [6]. As it carries significant structural information regarding the connectivity of the graph it forms the basis of spectral graph partitioning heuristics, see, e.g. [12] for a review. The Fiedler vector of $\mathbf{L}$ is used to produce a bipartition of the graph such that those vertices that have negative values in the eigenvector form one side of the bipartition $G_{\mathbf{S}}$ and the vertices with positive values are the other side $G \backslash G_{\mathbf{S}}$. For motivation of the use of Fiedler vectors for graph bipartitioning consult e.g. [12].

3 Random Walks on Graphs

Most of the information included in this section comes from the survey of Lovász [11].

Given a graph G we can define simple random walk on G. Consider a random walker, who initially starts at the origin vertex j. At each step the walker picks uniformly at random one neighboring vertex i and moves to it with the following probability:

$$P(j,i) = \begin{cases} \frac{1}{d(j)} & \text{if } j \sim i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Let A and B be two vertices of the graph G. The probability $p(i)$ that a random walker starting at node i reaches the node A before it reaches node B equals:

$$p(i) = \sum_{j \sim i} \frac{1}{d_i} p(j) \tag{2}$$

If we know the probability $p(j)$ that each of the neighbors of node i sent a random walker to A before B then $p(i)$ is an average of these probabilities. One can see a striking similarity between the probability vector p and the Fiedler vector. Each Laplacian eigenvalue λ and the corresponding eigenvector $\mathbf{f}$ fulfil the following equation:

$$\mathbf{Df} - \lambda \mathbf{f} = \mathbf{Sf} \tag{3}$$

and after transformation:

$$(d_i - \lambda) f_i = \sum_{j \sim i} f_j \tag{4}$$

As the second Laplacian eigenvalue is usually mach smaller than the degree of a vertex, we receive the following approximation for the i-th component of the Fiedler vector:

$$f_i \approx \frac{\sum_{j \sim i} f_j}{d_i} \tag{5}$$

Taking into consideration formulas (2) and (5) we can see that the vector p can constitute a good approximation of the Fiedler vector, especially for components corresponding to vertices with high degrees. If the second eigenvalue of the Laplacian equals zero (as in a case of two or more groups) formula 5 gives exact values of the Fiedler vector components.

This version of the random walks has one major drawback. It does not always converge: consider for instance a bipartite one-dimensional graph; the walk started at a vertex on the left will continue hopping back and forth between left and right without ever converging to any distribution.

Since $\mathbf{W} = \mathbf{D}^{-1}\mathbf{S}$ is the natural random walk transition matrix associated with a connected, undirected graph G, it follows that:

$$\mathbf{Z} = \alpha\mathbf{I} + (1-\alpha)\mathbf{W} \tag{6}$$

represents one step of the α-lazy random walk transition matrix, in which at each step there is a holding probability $\alpha \in [0, 1]$. In a lazy random walk at time t the walker:

– takes a step of the original random walk with probability $1-\alpha$,
– stays at the current vertex with probability α.

The probability that in the step $t+1$ he is at the vertex i equals:

$$pr_{t+1}(i) = \alpha pr_t(i) + (1-\alpha)\sum_{i\sim j}\frac{1}{d_j}pr_t(j) \tag{7}$$

Regardless of starting distribution, lazy random walk always converges to stable distribution. In stable distribution, every vertex is visited with probability proportional to its degree:

$$\pi(j) = \frac{d(j)}{\sum_i d(i)}$$

The fact that $\mathbf{W}$ and $\mathbf{Z}$ are not symmetric matrices makes their analysis complicated. The normalized lazy random walk matrix is defined as:

$$\mathbf{Z_s} = \mathbf{D}^{1/2}\mathbf{Z}\mathbf{D}^{-1/2} = \alpha\mathrm{I} + (1-\alpha)\mathbf{D}^{-1/2}\mathbf{S}\mathbf{D}^{-1/2}$$

The matrices $\mathbf{Z_s}$ and $\mathbf{Z}$ have the same eigenvalues and related eigenvectors. If $\mathbf{Z}$ has eigenvalue μ_i with an eigenvector f_i, $\mathbf{Z_s}$ has the same eigenvalue with eigenvector $\mathbf{D}^{1/2}f$. Moreover the eigenvalues μ_i of the matrices $\mathbf{Z}$ and $\mathbf{Z_s}$ are related with the eigenvalues ω of the matrix $\mathbf{W}$ in the following way: $\mu_i = \alpha + (1-\alpha)\omega_i$. On the other hand matrices $\mathbf{W}$ and $\mathbf{L_n}$ have the same eigenvalues.

Let pr_0 be an arbitrary initial distribution, and pr_t be the distribution after t steps of the lazy random walk. Then the rate of convergence of the lazy random walk to the stationary distribution π [11],

$$\|pr_t - \pi\|_2 \leq (1-\lambda)^t \cdot \sqrt{\frac{\max_j d(j)}{\min_i d(i)}} \tag{8}$$

where λ is the spectral gap defined as the difference between the first and the second eigenvalues of $\mathbf{Z}$: $\lambda = \mu_1 - \mu_2$. The eigenvalues of $\mathbf{W}$ lie in $[-1, 1]$, and thus those of $\mathbf{Z}$ lie in $[-1+2\alpha, 1]$. We can think of α as a constant in $[0, 1]$, so the smallest eigenvalue is bounded away from -1. The largest eigenvalue is still 1. For any connected graph, the gap is simply:

$$\lambda = (1-\alpha)(\omega_1 - \omega_2) \tag{9}$$

Then for any $\epsilon > 0$ the number of steps t_ϵ for the lazy random walk distribution p_t to be within ϵ of the stationary distribution π is $O(\frac{1}{\lambda(G)} \log \frac{\bar{d}(G)}{\epsilon})$, where $\bar{d}(G) = max_{j,i \in V} \frac{d(j)}{d(i)}$ is the degree of G.

4 Literature Overview

The new trend is to replace eigenvectors with vectors obtained by running random walks. As it was showed in Sect. 3 the second eigenvector of the Laplacian can be approximately computed by iterating random walks. In a case of early stopping of distribution evaluation one does not obtain an eigenvector, but either an approximate eigenvector or a locally-biased analogue of the leading eigenvector. An important aspect of replacing an eigenvector with a random walk early distribution vector is its robustness. Mahoney [13] explained the idea on the ground of machine learning. Let us suppose that there is a "ground truth" graph that we want to partition, but the graph that we actually have available to compute with, is a noisy version of this ground truth graph. So, if we want to compute the leading nontrivial eigenvector of the unseen graph, then computing the leading nontrivial eigenvector of the observed graph is in general not a good idea. The reason is that it can be very sensitive to noise, e.g., mistakes or noise in the edges. On the other hand, if we perform a random walk and keep the random walk vector, then that is a better estimate of the ground truth eigendirection. The idea is that eigenvectors are unstable but random walks are not unstable.

One of the first early-stopped solution presented Wu and Huberman [21] in their clustering algorithm. They applied an electrical circuit analogue in order to reveal a graph structure. They imagined each edge to be a resistor with the same resistance, and connected a battery between A and B so that they have fixed electrical potentials, eg. 1 and 0. Vertices A and B belong to different clusters. Having made these assumptions the graph can be viewed as an electric circuit and current flows through each edge (resistor) as random walks. By solving Kirchhoff equations they obtained the potential value of each node. Using this information it is possible to partition the graph into two parts. A node belongs to G_1 if its electrical potential is greater than a certain threshold, and it belongs to G_2 if its potential is less than that threshold. Let us assume that C connects to n neighbors $D_1; \ldots; D_n$. According to Kirchhoff equation the total current flowing into vertex C should sum up to zero:

$$\sum_{i=1}^{n} I_i = \sum_{i=1}^{n} \frac{V_{D_i} - V_C}{R} = 0$$

where I_i is the current flowing from D_i to C, and V_C and V_{D_i} are potentials of the appropriate nodes. Thus

$$V_C = \frac{1}{n} \sum_{i=1}^{n} V_{D_i}.$$

That is, the potential of a node is the average of the potentials of its neighbors and can be computed as:

$$V_i = \frac{1}{d_i} \sum_{(i,j) \in E} V_j = \frac{1}{d_i} \sum_{j \in G} V_j s_{ij}$$

where d_i is the degree of node i and s_{ij} is the affinity matrix of the graph. There is a straight analogy between random walks and electric network. The voltage at an arbitrary vertex i equals the probability of reaching A from i before reaching B (Eq. 2).

Starting from the node with potential 1, they consecutively updated a node's potential to the average potential of its neighbors. Repeating the updating process for a finite number of rounds, one reaches an approximate solution within a certain precision, which depends on the number of iteration rounds. In other words, the obtained precision depends on number of repeated rounds and not on the size of the graph, so the total computational cost is always $O(n + m)$. Although the solution is very fast it demands answering a critical question: "How to pick the two poles so that they lie in different communities?" The authors proposed some heuristical methods to solve the problem, but they increased the computational complexity of the algorithm.

The `MCL` algorithm [4] simulates random walks within a graph by alternation of two operators called expansion and inflation. The first one coincides with taking the power of a stochastic matrix using the normal matrix product (i.e. matrix squaring). Inflation corresponds with taking powers entrywise of the matrix, followed by a scaling step, so that the matrix elements (on each column) correspond to probability values. The goal of the last procedure is to favor same paths inside one cluster over others leading to different clusters. Its computational complexity is of $O(md^2)$, where d represents an average number of nonzero elements in one column of the stochastic matrix.

Mathematically motivated approach showed Spielman and Teng, who introduced a local partitioning algorithm with a remarkable approximation guarantee and bound on its computational complexity [17,18]. Their algorithm has a bounded work/volume ratio, which is the ratio between the work performed by the algorithm on a given run (meaning the number of operations or computational complexity), and the volume of the set it outputs. Their algorithm computes a sequence of vectors that approximate the sequence of probability distributions of a random walk from the starting vertex. The support of these vectors is kept small by removing tiny amounts of probability mass at each step.

Their most recent algorithm uses graph sparsification [19], that is the task of approximating a graph by a sparse graph. They introduced a new notion of spectral sparsification. A spectral sparsifier is a subgraph of the original whose Laplacian quadratic form is approximately the same as that of the original graph on all real vector inputs. By applying the method inside the inverse power method, they computed approximate Fiedler vectors in time $O(mlog^c m)$, where m is the number of edges in the original graph and c is some absolute constant.

Andersen, Chung, and Lang developed an improved version of the Spielman and Teng's algorithm for computing approximate PageRank vectors [1]. Instead of computing a sequence of vectors p^{t+1} they found personalized PageRank vector, which simplifies the process of finding cuts and allows greater flexibility when computing approximations. Their method allows us to find cuts using approximations with larger amounts of error, which improves the running time.

5 The ESLRW Algorithm

We have modified the method of Wu and Huberman, and according to the α-lazy random walk process, propose the following formula, describing the evolution of the distribution $p(t)$ at the vertex i:

$$p_{t+1}(i) = \alpha p_t(i) + (1-\alpha)\frac{1}{d_i} \cdot \sum_{(i,j)\in E} p_t(j) \tag{10}$$

Our intention was to trace the evolution of a probability distribution propagation on graph and use it for graph bipartitioning. We have focussed on the beginning stages of the process in order to observe the system before it reaches a stationary state. This way we kept a reasonably small computational cost and solution simplicity. We used the lazy random walk instead of the ordinary random walk to avoid cycling of the distribution values. Moreover we can see from equation (8) that the rate of convergence grows with increasing value of spectral gap. Taking into consideration formula (9) we decided to use α smaller then 0.5, which is the standard value for lazy random walks. Although our purpose was not to reach the convergence we just wanted to make the process a bit faster. In our experiments $\alpha = 0.3$.

Our algorithm differs from the Wu-Huberman's solution. There is no necessity to indicate two distant vertices A and B belonging to different clusters, with p values one and zero, respectively. The vertex with the highest degree was chosen as the seed vertex A, in order to make the process faster. As our algorithm runs for a small number of iterations, there are always vertices with p value equal zero, so one of them can constitute vertex B to meet the conditions of Eq. 2.

The evolution of the distribution probability is stopped when the maximum change of the vector component $p(i)$, $i \in V$, is smaller than an established threshold θ. We did not take into consideration the change of the whole vector but only of the components that fluctuate considerably during the process of evaluation. This heuristics is motivated by fact that the probability changes in the subgraph of the seed vertex are larger than in the rest of the graph. Between two not very well separated subgraphs there is a bottleneck that allows for spreading the walk, but it significantly slows down mixing. The bottleneck makes the other part of the graph difficult to reach from the location of the seed and limits the speed of convergence.

It is clear that probability distribution of random walk in the subgraph containing the seed vertex differs from those in the other subgraphs. In order to extract vertices belonging to the same subgraph as the seed vertex, we looked for the largest gap between probability values at all vertices and divide the whole set at the gap.

In our algorithm it is no necessity to give the number of clusters. We used the modularity function, a well known partitioning quality measure introduced by Newman and Girvan [9], in order to decide whether to bipartition the graph farther. After the first bipartitioning we checked with the help of the modularity, whether the next division gives better clustering. So the next stage of the ESLRW algorithm involves calculation of the modularity function.

According to Newman a good division of a graph into partitions is not merely one in which there are few edges between groups; it is one in which there are fewer than expected edges between groups. The modularity Q is, up to a multiplicative constant, the number of edges falling within groups minus the expected number in an equivalent graph with edges placed at random, or in functional form:

$$Q = \frac{1}{2m} \sum_{i,j} \left[a_{ij} - \frac{d_i d_j}{2m} \right] \delta(g_i, g_j)$$

where $\delta(g_i, g_j) = 1$ if $g_i = g_j$ and 0 otherwise, g_i stands for a group of the vertex i, and m is the number of edges in the graph. Although modularity is widely used to evaluate the cluster structure of a graph, it is prompt to increase as the number of divisions increases, including even within-cluster divisions. We assumed that a sequential division of a graph makes sense if Q increases by ten percentage of the previous modularity value. It avoids unnecessary growth of the number of clusters.

If cutting a set of vertices through a selected gap would not cause a reasonable gain of modularity, partitioning of the set is not executed and the set is labeled as complete. The algorithm finishes if all its vertices belong to complete sets.

To sum up our algorithm consists of the following steps:

- Given a graph construct its adjacency matrix
- Calculate the vector p
- Use the vector p to bipartition the graph
- Decide if the current group should be sub-divided, and recursively repartition the segmented parts if necessary.

To create the matrix $\mathbf{S}$ the k-nearest neighbors were determined for each point i on the basis of the Euclidean distance. The adjacency matrix was obtained with the help of a mutual k-nearest neighbor graph. The graph was constructed by connecting i to j if i is among the k-nearest neighbors of j and vice versa. If two vertices are connected, the appropriate value in the adjacency matrix $\mathbf{A}$ equals 1, otherwise it is 0. The parameter k was chosen experimentally.

After constructing the adjacency matrix of the graph, the vector p is calculated according to formula (10). Values of the p vector depend on the starting point of the random walk. If the starting point is located in the middle of the group and does not have connections with vertices belonging to other groups the calculation is fast and partitioning exact. To make our algorithm as simple as possible we resign from additional steps and decide to choose always the vertex with the maximum degree.

Computational complexity of the bipartitioning depends on one round complexity and the number of iterations. The round involves updating distribution probabilities at each vertex and is proportional to the number of vertices n. The random walk is executed till the maximum value, by which the distribution at one vertex is updated, exceeds a given threshold. Additionally we limited the maximum number of iterations to $itmax = 100$. To sum up the computational complexity of bipartitioning in our algorithm is $O(n \cdot it)$, where it stands for the number of iterations and is smaller than $itmax$. Taking into consideration the fact that in our experiments we used graphs with the number of nodes varying between 3000 and 20000, the number of iterations is comparably small. We can say that our solution almost linearly depends on the size of the graph. As the procedure is recursively repeated till all the clusters are found and all the vertices labeled as complete, the whole complexity of the `ESLRW` algorithm is $O(n \cdot it \cdot k)$, where k represents the number of clusters.

6 Experimental Results

We compared the performance of the `ESLRW` algorithm (implemented in MATLAB) to other clustering algorithms: `NJW` [15], K-means, K-means-based Nyström spectral clustering (K-NASP) [23], and `LI-ASP` [3]. The first two are popular standard algorithms and the `NJW` uses exact values of symmetric Laplacian eigenvectors. The last algorithm is a new approximate solution in which two improvements were made comparing to traditional spectral clustering. First, a sparse affinity graph was adopted to improve the performance of spectral clustering on a small representative dataset. Second, local interpolation was utilized to improve the extension of the clustering result. The main computation of the `LI-ASP` includes three parts: obtaining the representative points, running spectral clustering on the representative points and extending the clustering result by the local interpolation. The computational complexity of this solution is $O(p^3) + O(n \cdot m \cdot p \cdot k^2)$, where p stands for the number of representatives, $p \ll n$. K-means-based Nyström spectral clustering `K-NASP` uses Nyström law rank matrix approximation with the landmark points chosen as the K-means cluster centers. This method allows to extrapolate the complete grouping solution using only a small number of samples. The computational complexity of the algorithm is $O(m \cdot n)$.

We conducted experiments with several real-world large datasets of various sizes, described as follows:

– MNIST [20] consisting of handwritten digits, with a total of 70,000 examples; each example is a 28×28 gray-level image, and the dimensionality is 784. Considering the MATLAB memory limitation, we chose the first 2000 examples in the training set of each digit in our experiments. In addition, we also constructed two subsets of MNIST, MNIST358 and MNIST1479. MNIST358 consists of the examples of digits 3, 5, and 8, and MNIST1479 consists of the examples of digits 1, 4, 7, and 9.

- USPS [10], the US Postal Service (USPS) handwritten digit dataset, in which each sample is a 16×16 image. It contains ten digits 0–9. Two subsets of USPS similar to those of MNIST were also formed, which are denoted by USPS358 and USPS1479.
- Pendigit [2], a handwritten digit data set consisting of 250 samples from 44 writers. In our experiments the training and testing sets were used as two datasets, which are denoted by Pendigit train and Pendigit test, respectively.
- LetterRec [2] containing the features of 26 capital letters of the English alphabet. The character images are based on 20 different fonts. They are randomly distorted.

Table 1 summarizes basic information of these datasets. It is worth noting that the number of the instances in each dataset is more than 2000.

All the datasets are labeled, which enables evaluation of the clustering results against the labels using the accuracy of clustering (ACC) and normalized mutual information (NMI), as measures of division quality. For both measures higher number means better partitioning. We refer an interested reader to [14] for details regarding the measures. The same benchmark datasets were applied in experiments for the `LI-ASP` algorithm by Cao *et al.* We also used their results.

Table 1. A summary of datasets.

Dataset	♯ of instances	♯ of features	♯ of classes
MNIST	20000	784	10
MNIST358	6000	784	3
MNIST1479	8000	784	4
USPS	9298	256	10
USPS358	2357	256	3
USPS1479	3919	256	4
Pendigit train	7494	16	10
Pendigit test	3492	16	10
LetterRec	20000	16	26

The performance of the algorithms show Tables 2 and 3. Not all results are presented because the datasets of 20000 points were too large for `NJW` MATLAB implementation. We can see the superiority of the `ESLRW` algorithm over the other tested solutions. The results prove that the presented method is competitive to the other solutions in terms of the quality of partitioning, measured with the help of the accuracy of clustering and the NMI quality measure.

We have also applied the `ESLRW` algorithm to find protein complexes. In our experiments we used two well known yeast PPI datasets. The first dataset

Table 2. Comparison of K-MEANS, NJW, K-NASP, LI-ASP, and ESLRW algorithms in terms of the accuracy of clustering.

Algorithm	K-MEANS	NJW	K-NASP	LI-ASP	ESLRW
MNIST	0.4912	/	0.5284	0.6999	**0.7403**
MNIST358	0.1357	**0.7548**	0.5781	0.7175	0.7220
MNIST1479	0.3502	0.3623	0.4446	0.5032	**0.5449**
USPS	0.5921	0.6562	0.6319	0.6746	**0.9420**
USPS358	0.7170	0.9163	0.9189	0.9606	**0.9845**
USPS1479	0.5214	0.9615	0.5816	0.9446	**0.9770**
Pendigit test	0.6710	**0.8263**	0.6306	0.7705	0.8210
Pendigit train	0.6686	0.7889	0.6955	0.8113	**0.8193**
LettRec	**0.3631**	/	0.2728	0.3197	0.324

Table 3. Comparison of K-MEANS, NJW, K-NASP, LI-ASP, and ESLRW algorithms in terms of the NMI.

Algorithm	K-MEANS	NJW	K-NASP	LI-ASP	ESLRW
MNIST	0.4912	/	0.5102	0.7138	**0.7758**
MNIST358	0.1357	0.4670	0.1284	0.4284	**0.6008**
MNIST1479	0.3502	0.4700	0.3015	0.4365	**0.6491**
USPS	0.5921	0.7723	0.6178	0.7347	**0.8809**
USPS358	0.7170	0.9163	0.7224	0.8387	**0.9187**
USPS1479	0.5214	0.8662	0.5747	0.8506	**0.9066**
Pendigit test	0.6710	0.8263	0.7198	0.7793	**0.8646**
Pendigit train	0.6686	0.8008	0.7388	0.7949	**0.8623**
LettRec	0.3631	/	0.3752	0.4315	**0.4460**

(PPI-D1) is prepared by Gavin *et al.* [8] and the second dataset (PPI-D2) is a combined PPI dataset containing yeast protein interactions generated by six individual experiments [22]. The first one consists of 990 and the second one of 1440 proteins. We compared our solution with the a standard algorithm used for protein networks - the `MCL` algorithm. Our method outperformed the other algorithm in terms of modularity measure. For the first set PPI-D1 modularity of our algorithm was 0.8157 and of the other one 0.7692, and for the second set PPI-D2 respectively 0.8134 and 0.7835.

7 Conclusions

The computational complexity of the presented algorithm is reasonably small. It depends on the number of iterations, that never exceeds 100. The number constitutes only small percentage of iterations needed for convergence. For example in case of the Pendigit test set, the number of iterations to achieve the accuracy of 0.001 is of $3.5 \cdot 10^3$ whereas in our algorithm the early stop was made after 75 rounds.

We have presented a fast algorithm that uses random walk procedure for graph bipartitioning. The solution uses a close relation between the Fiedler vector and the vector, whose components are probabilities of reaching a seed vertex before the other distant vertex starting from i. We calculated the local rough approximation of the Fiedler vector. The vector was obtained after considerably small number of algorithm iterations. The algorithm gives promising results despite its simplicity. Our experiments show superiority of graph partitioning with the use of the local approximation of the Fiedler vector over cuts resulting from the vector itself. Because of small computational complexity our method seems to be adequate for discovering structures in large graphs.

References

1. Andersen, R., Chung, F.R.K., Lang, K.J.: Local graph partitioning using PageRank vectors. In: FOCS 2006, pp. 475–486 (2006)
2. Bache, K., Lichman, M.: UCI Machine Learning Repository (2013). http://archive.ics.uci.edu/ml
3. Cao, J.Z., Chen, P., Dai, Q., Ling, B.W.K.: Local information-based fast approximate spectral clustering. Pattern Recogn. Lett. **38**, 63–69 (2014)
4. van Dongen, S.: Graph clustering via a discrete uncoupling process. SIAM J. Matrix Anal. Appl. **30**(1), 121–141 (2008)
5. Enright, A.J., van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res. **30**(7), 1575–1584 (2002)
6. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Math. J. **23**(98), 298–305 (1973)
7. Flake, G., Lawrence, S., Lee Giles, C., Coetzee, F.: Self-organization and identification of Web communities. IEEE Comput. **35**(3), 66–71 (2002)
8. Gavin, A.C., et al.: Functional organization of the yeast protein by systematic analysis of protein complexes. Nature **415**, 141–147 (2002)
9. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA **99**(12), 7821–7826 (2002)
10. Hull, J.J.: A database for handwritten text recognition research. IEEE Trans. Pattern Anal. Mach. Intell. **16**, 550–554 (1994)
11. Lováasz, L.: Random walks on graphs: a survey, combinatorics, Paul Erdös is Eighty 2, pp. 146 (1993)
12. von Luxburg, U.: A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
13. Mahoney, M., Orecchia, L.: A local spectral method for graphs: with applications to improving graph partitions and exploring data graphs locally. J. Mach. Learn. Res. **13**, 2339–2365 (2012)
14. Manning, C., Raghavan, P., Schtauze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
15. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: analysis and an algorithm. Adv. Neural Inf. Process. Syst. **14**, 849–856 (2001)

16. Shi, J., Malik, J.: Normalized cuts and image segmentation. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 1997), pp. 731–752. IEEE Computer Society (1997)
17. Spielman, D.A., Teng, S.-H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: STOC 2004, pp. 81–90. ACM, New York (2004)
18. Spielman, D.A., Teng, S.-H.: A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. CoRR, abs/0809.3232 (2008)
19. Spielman, D.A., Teng, S.-H.: Spectral sparsification of graphs. SIAM J. Comput. **40**, 18–025 (2011)
20. Yann, L., Corinna, C.: The MNIST database of handwritten digits (2009). http://yannlecun.com/exdb/mnist/
21. Wu, F., Huberman, B.A.: Finding communities in linear time: a physics approach. Eur. Phys. J. B **38**(2), 331–338 (2004)
22. Zaki, N.M., Lazarova-Molnar, S., El-Hajj, W., Campbell, P.: Protein-protein interaction based on pairwise similarity. BMC Bioinf. **10**, 1–12 (2009)
23. Zhang, K., Kwok, J.: Improved Nyström low rank approximation and error analysis. In: Proceedings of the International Conference on Machine Learning (ICML) (2008)
24. Zhou, H., Lipowsky, R.: Dynamic pattern evolution on scale-free networks. Proc. Nat. Acad. Sci. USA **102**(29), 10052–10057 (2005)