Accurate Spectral Clustering for Community Detection in MapReduce

Serafeim Tsironis
Athens University of Economics and Business
Athens, Greece
tsironis@telecom-paristech.fr

Mauro Sozio
Telecom ParisTech
Paris, France
sozio@telecom-paristech.fr

Michalis Vazirgiannis
LIX - École Poltechnique
Palaiseau Cedex, France
mvazirg@lix.polytechnique.fr

Abstract

Spectral clustering has become one of the most popular clustering algorithms and it is currently being used in a wide range of applications. Unfortunately, the running time of spectral clustering algorithms might be cubic on the size of the input dataset, which makes it prohibitive to use this approach on very large datasets. In recent years, several efforts have been made to cope with these scalability issues, however, a satisfactory solution is still missing. In this work [1], we investigate a variant of the spectral clustering which can be efficiently parallelized in MapReduce and we study its effectiveness in finding communities on large-scale social networks. Our evaluation on both real and synthetic large-scale social networks shows promising results for our approach.

1 Introduction

Clustering is one of the most important subroutine in tasks of machine learning and data mining. Spectral clustering, which exploit pairwise similarities of data instances, has been widely used in several areas such as image segmentation and community detection, because of its effectiveness to find clusters.

However, spectral clustering algorithms are not efficient as their running time is cubic in the size of the input dataset. This makes these algorithms incapable of handling large graphs. MapReduce is a programming model for large-scale data processing and is used for making distributed calculations in a cluster of computers.

In this work, we investigate a variant of the spectral clustering which can be efficiently parallelized in MapReduce and we study its effectiveness in finding communities on large-scale social networks. Our approach consists of computing efficiently an eigenvalue decomposition and then to use a recent parallel version of k-means++ (called k-means|| [26]) which comes with approximation guarantees on the quality of the clustering so computed. We then focus on the problem of finding communities in large-scale social networks. We conduct an experimental evaluation on both synthetic and real-world social networks which shows promising results.

The remainder of this paper is organized as follows: in section 2, we present the basic principles of spectral clustering algorithms and the bottlenecks for handling large graph datasets. In section 3,

[1] This work was a part of master thesis of S. Tsironis as an intern student, under the collaboration of Telecom ParisTech and Ecole Polytechnique.

we show the related work on spectral clustering approaches for large datasets and we propose some solutions to this problem. In section 4, we present our parallel spectral clustering algorithm and we mark some technical issues and our contributions to the problem. Experimental results in section 5 show the evaluation of our approach on both real-world and synthetic large scale social networks. Section 6 shows our concluding remarks.

2 Background

This section presents the spectral clustering algorithm and describes the bottlenecks that led to the need of a parallel approach, in order to deal with massive graphs.

2.1 Basic notations

Given a set of n points x_1, ..., x_n, spectral clustering algorithms constructs an affinity matrix $A \in R^{nxn}$ where $A_{ij} \geq 0$ represents the similarity between x_i and x_j. We consider a commonly used spectral clustering algorithm, proposed by Ng et al. [1], called *normalized* spectral clustering. The similarity information is then used to group these points into k clusters. The most common similarity function used is the Gaussian:

$$S_{ij} = exp\left(\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

In our implementation, we use the similarity representation where:

$$A_{ij} = \begin{cases} 1 & \text{if } i,j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$

After forming the affinity matrix, spectral clustering algorithm constructs the graph Laplacian [2] matrix, that constitutes a graph representation of the initial data points. There are several forms of the Laplacian matrix [3], but we consider the normalized symmetric Laplacian:

$$L_{sym} = I - D^{-1/2} A D^{-1/2},$$

where D is the diagonal matrix with

$$D_{ii} = \sum_{j=1}^{n} A_{ij}$$

To implement spectral clustering one has to compute the first k eigenvectors (those corresponding to the k smallest eigenvalues) of graph Laplacian matrix, which is usually sparse. Eigenvalue decomposition of the graph Laplacian is the most important step of spectral clustering algorithm, because it obtains a representation of the initial data into a low-dimensional space in order to be clustered.

Let $U \in R^{nxk}$ be the matrix contains the first k eigenvectors u_i,..., u_k as columns. Spectral clustering algorithm forms a matrix $T \in R^{nxk}$ from U by normalizing the rows to norm 1, that is set $T_{ij} = U_{ij}/(\sum_{k=1}^{j} U_{ik}^2)^{-1/2}$. Each row of matrix T is a vector $(y_i)_{,i=1...n}$ that represents an initial data point into the new low-dimensional space. Finally, the points y_i are clustered into clusters C_1,...,C_k using k-means clustering algorithm.

2.2 Eigenvalue decomposition

Methods for sparse eigenproblems [4]-[5] usually obtain the solution from the information generated by the application of the matrix to various vectors. Matrices are only involved in matrix-vector products. This preserves sparsity and also allows the solution of problems in which matrices are not available explicitly.

Lanczos algorithm [6], the most well-known large scale eigensolver, produces an orthogonal transformation of a symmetric matrix A into a tridiagonal matrix T. The point of the algorithm is that, due to the orthogonality of the transformation, matrix T is similar to the original matrix A (for instance, each eigenvalue of T is also an eigenvalue of A), but it is much easier to calculate with T because it has a tridiagonal form. This idea of Lanczos is frequently applied to solve eigenvalue problems and linear systems.

2.3 K-means

K-means algorithm [7] is applied to objects that are represented by points in a d-dimensional vector space. Thus, it clusters a set of d-dimensional vectors, $D = \{x_i | i = 1, ..., n\}$, where $x_i \in R^d$ denotes the i-th object or "data point". K-means algorithm clusters all of the data points in D such that each point x_i falls in one and only one of the k partitions.

In clustering algorithms, points are grouped by some notion of "closeness" or "similarity". In k-means there is a set of cluster representatives $C = \{c_i | i = 1, ..., k\}$, which are called the cluster means or the cluster centroids. K-means attempts to minimize the total squared Euclidean distance between each point x_i and its closest cluster centroid c_j.

2.4 Complexity analysis

Regarding the computational complexity of spectral clustering algorithm, the most expensive step is the computation of the eigenvalues/eigenvectors of Laplacian matrix. This process has time complexity O(n^3), where n is the number of input data points. The construction of similarity matrix has time complexity O(n^2) and the application of k-means in the results of eigenvalue decomposition costs O($nldk$), where n is the number of input data points, l is the number of k-means iterations, d is the dimensionality of the input data and k is the number of final clusters.

Despite the importance of spectral clustering algorithms, they are not widely be viewed as a competitor to classical algorithms as hierarchical clustering and k-means for large scale data mining problems. The overall computational complexity of spectral clustering algorithm is O(n^3). This makes spectral clustering methods becoming infeasible for problems with n on the order of thousands. Furthermore, problems with n in the order of millions (or billions) are entirely out of reach.

3 Related work

Research on speeding up the execution of Spectral Clustering algorithms has been focused mainly on approximating the affinity matrix using nearest neighbour techniques [8]-[9] or by extracting dense regions in large graphs [9]-[11] or techniques that can handle large number of edges in one machine [12]. Another important field, that has an important role in the total running time and complexity of Spectral Clustering, is the eigenvalue decomposition step. Several methods [8], [13]-[14] have been used to approximate the eigenvectors of the affinity matrix.

In MapReduce [15], existing approaches still focus on the preprocessing step of the input data points in order to speed up the whole process. Gao et al. [16], proposed a distributed spectral clustering approach that focuses on preprocessing the input data points using Local Sensitivity Hashing (LSH) and applying Mahout implementation for Spectral Clustering on the results of LSH. Cordeiro et al. [17], proposed a hybrid method for clustering, that minimizes I/O cost and network cost among processing nodes. This method includes a parallel clustering method (ParC method) and a sample-and-ignore method (SnI method).

Our approach focuses in two directions: efficient selection of top k eigenvectors of the Laplacian matrix and careful selection of the initial centroids for k-means step. Our implementation makes use of the advantages of MapReduce and provides a spectral clustering method that can handle large graphs in a reasonable time.

3.1 Eigensolver selection

Taking into account the complexity analysis of spectral clustering algorithms, it is completely infeasible to get the exact eigenvalues/eigenvectors of a large matrix and especially using a centralized approach. There are many parallel large scale eigensolvers [18]-[20] proposed during the last years. Most of these algorithms are adaptations of power method to find eigenpairs of a square matrix.

HEIGEN, proposed by Faloutsos et al. [21], as part of the peta-scale graph mining library PEGASUS [22]-[23], is a parallel eigensolver designed to be accurate, efficient, and able to run on the MapReduce environment. MapReduce enables HEIGEN to handle matrices much larger that these that can be handled by algorithms based on MPI.

HEIGEN uses the Lanczos eigensolver for symmetric matrices, and especially an improved version of basic Lanczos algorithm, the Lanczos-SO (Selective Orthogonalization). The purpose of selective orthogonalization is to prevent the computation of many unwanted copies of all the well-separated outer eigenvectors. This reduces the number of Lanczos steps required to compute the wanted eigenvalues and eigenvectors and so keeps the number of calls on the input matrix as low as possible.

3.2 K-means selection

K-Means is an algorithm that can be easily parallelized in MapReduce in order to deal with large number of data points [24]. Nevertheless, k-means may need a large number of iterations in order to converge to a solution. MapReduce is not suitable for executing algorithms with large number of iterations.

In order to overcome this issue, it is important to make a careful choice of the initial cluster centroids which lead to a good solution and will converge after a small number of iterations. Arthur and Vassilvitskii [25], proposed k-means++ that selects carefully a set of centroids that leads k-means to converge quickly to a solution but the problem for large datasets is that it may perform many passes over the input data, especially when we need a large number of clusters. This problem solved by applying a sampling method [26], that select a subset from the input data points and them extract the initial centroids using k-means++ algorithm.

4 Proposed method

In this section we discuss the technical issues that contribute in efficiency of our algorithm and finally we present our approach that can perform spectral clustering for large graph datasets.

4.1 Technical issues

HEIGEN improves the efficiency of eigenvalues/eigenvectors computation by adapting the use of block-based operations [21]. Using blocks the number of the intermediate keys (and the number of reducers in use) for MapReduce are fewer than operations that do not use blocks and the performance of the reducers is increased. A typical example of an operation that is optimized using blocks, is the matrix-vector multiplication, where the matrix is large and the vector is large and dense.

Matrix operations, like matrix-vector and matrix-matrix multiplication, can also be applied in matrix/vectors that can be hold in the main memory of a single machine. In this case, the matrix/vector that fits into main memory can become available to all mappers at the same time using distributed cache functionality in Hadoop. This functionality decreases the number of MapReduce jobs that have to be executed and, furthermore, decreases the execution time for these operations.

4.2 Algorithm

Let G = (V, E) an undirected unweighted graph of a set of data points and A is the affinity matrix, representing the similarity between all nodes inside this graph. The next figure shows the proposed spectral clustering algorithm, that can deal with massive graph datasets.

In comparison with the most well-known spectral clustering algorithms there is a difference in steps 3-4. More specifically, spectral clustering algorithms compute the smallest k eigenvalues with the corresponding eigenvectors of the normalized symmetric Laplacian matrix $L_{sym} = I - D^{-1/2}AD^{-1/2}$. Our approach is lead by the fact that matrices L_{sym} and $D^{-1/2}AD^{-1/2}$ have in total the same eigenvectors, with the difference that the eigenvectors of L_{sym} corresponding to the smallest eigenvalues are the same with the eigenvectors of $D^{-1/2}AD-1/2$ corresponding to the largest eigenvalues. For the eigenvalues, it holds that if λ is an eigenvalue of L_{sym} then $(1 - \lambda)$ is an eigenvalue of $D^{-1/2}AD^{-1/2}$.

Our objective is to use this approach for detect communities in large networks, and especially in social networks. Our contributions are led by the fact that we want to perform community detection efficiently and with accurate results. HEIGEN and k-means|| are two approaches that speed up the spectral clustering algorithm and provide good clustering results for large networks.

Algorithm 1 Proposed Parallel Spectral Clustering Algorithm

1: Construct diagonal matrix D, where the element (i, i) in the main diagonal is the degree of the i-th node in the graph.
2: Compute inverse square root matrix $D^{-1/2}$ of the degree matrix D.
3: Compute matrix $L = D^{-1/2}AD^{-1/2}$
4: Compute the eigenvectors of matrix L, corresponding to the largest k eigenvalues of matrix L using HEIGEN algorithm.
5: Let U_{nxk} a matrix which contains the selected eigenvectors of matrix L as columns.
6: Form matrix $T \in R^{nxk}$ from U by normalizing the rows to norm 1, that is set $T_{ij} = U_{ij}/(\sum_{k=1}^{j} U_{ik}^2)^{-1/2}$.
7: Let $(y_i)_{,i=1...n}$ be the vector corresponding to the i-th row of T.
8: Cluster points y_i in R^k into clusters C_1,...,C_k using k-means|| algorithm.

Table 1: Synthetic graph datasets

DATASET	NODES	EDGES	GROUND TRUTH CLASSES
1	1000	24644	24
2	3000	148374	41
3	5000	251410	67
4	10000	982916	60
5	20000	1980228	100

5 Experiments

5.1 Datasets

We used both synthetic and real graph datasets in order to evaluate our method. Synthetic graph datasets were built using the Fortunato graph generator [27] and the size of the graph varies from 1000 to 20000. Table 1 shows the details for the synthetic datasets. The ground truth classes, created for each dataset, are disjoint to each other. The real graph dataset, taken from SNAP network (`http://snap.stanford.edu/data/`), is a product co-purchasing network collected by crawling Amazon website. The graph contains 334863 nodes and 925872 bidirected edges split into 151307 overlapping communities.

5.2 Results

We ran our experiments in a Hadoop clusters consisting of 5 machines, 1 master and 4 slave nodes. We set the maximum number of iterations for HEIGEN to 20 and the maximum number of k-means iterations to 30. For the centralized approach we implemented the spectral clustering algorithm, proposed by Ng et al. [1], using the Jama library implementation for eigenvalue decomposition and matrix operations and the WEKA implementation for k-means step.

Table 2 shows the evaluation of clustering results, produced by our parallel spectral clustering approach, in comparison with them produced by centralized spectral clustering approach. For this evaluation , we use the F-measure metric which combines the precision and recall ideas from Information Retrieval literature. Parallel spectral clustering approach gives better or equivalent clustering results with centralized approach when size is relatively small.

Regarding the running time of these two approaches, as we can see in Table 3, centralized approach is more efficient for smaller datasets, but for larger datasets it starts becoming infeasible to get a solution in a reasonable time. On the other hand, despite the fact that our parallel approach gives less accurate results as the size increases, it still gives good results in a reasonable time.

Table 2: F-measure results (Centralized vs. Parallel Spectral Clustering)

DATASET	CLUSTERS	EIGENVECTORS	F-MEASURE (CENTRALIZED)	F-MEASURE (PARALLEL)
1	20	6	0.7603	0.8354
2	40	4	0.7679	0.8113
3	60	5	0.8356	0.8209
4	60	5	0.7593	0.7416
5	100	5	0.8248	0.6678

Table 3: Running time (Centralized vs. Parallel Spectral Clustering)

DATASET	TIME (CENTRALIZED)	TIME (PARALLEL)
1	$<$1 min	$\sim$ 2 hours
2	$\sim$ 9 min	$\sim$ 3 hours
3	$\sim$ 43 min	$\sim$ 4 hours
4	$>$5 hours	$\sim$ 4.5 hours
5	$>$2 days	$\sim$ 5 hours

Furthermore, we ran these two implementation into a synthetic graph with 60000 nodes and almost 4 million edges where centralized spectral clustering approach failed to execute while parallel spectral clustering approach executed successfully.

The following figures show the evaluation of the clustering results, produced by our parallel spectral clustering approach, for a real graph dataset. We set the maximum number of iterations for HEIGEN to 100 and we used the top 51 eigenvectors. We extracted 100 clusters and the whole experiment completed successfully in almost 2 days.

F-measure is not a suitable metric when the available ground truth communities are overlapping. We decided to use normalized entropy in order to evaluate our results. As the ground truth communities are overlapping and our clusters disjoint, we compute the normalized entropy for every ground truth community in respect to the extracted clusters. The optimal value for the normalized entropy, where all data points inside a cluster belong also to the same community, is 0. On the other hand, the worst case for a cluster is every data point belong to a different cluster than the others.In this case, the value of the normalized entropy is 1.

As we can see in the figures, we split the normalized entropy into 11 classes and we map each community to a specific class in such a way that the normalized entropy of the community is less or equal the class value. Each plot in the figures shows the distribution of the communities in these classes and belong to a specific size range.

Figure 1 shows the normalized entropy class distribution for communities that contain at most 50 points. Our algorithm is performing very well for small communities as we can see in the first plot where more than 15000 communities have entropy value set to 0. We can see that many communities are concentrated into entropy classes with value less or equal to 0.6, which can be seen as an satisfying entropy value.

Figure 2 shows the normalized entropy class distribution for all communities. Because the majority of the communities have small size we split them into two plots, one that contains communities with size less or equal to 5000 and another one that contains the rest of them. In this figure, we can see clearly that more than 60000 communities have normalized entropy value set to 0, which is the optimal case.Also, it is shown that the majority of communities give a satisfying entropy value, even for those communities that have size more than 5000 points.

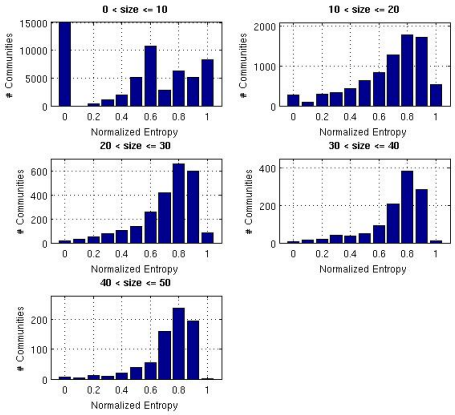

Figure 1: Normalized Entropy - Amazon Dataset (size <= 50)

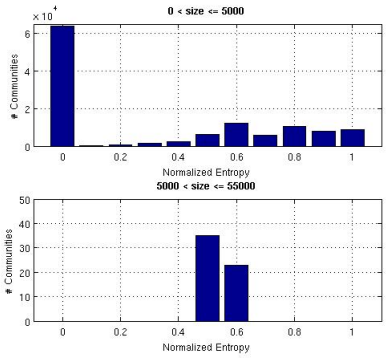

Figure 2: Normalized Entropy - Amazon Dataset (all)

6 Conclusion

In this paper, we have shown our parallel implementation of the spectral clustering to give equivalent results with proposed centralized approaches. Also we proved its ability to give accurate results for large datasets that one machine cannot handle. We plan to perform further experiments on larger graph datasets using larger MapReduce clusters. Because our approach supports undirected and unweighted graphs, we plan to extend its functionality to support also directed and weighted graphs.

References

[1] Ng, A.Y., Jordan, M.I, Weiss, J. (2001) *On Spectral Clustering: Analysis and an algorithm*. NIPS 2001:849-856.

[2] Huang, J. (2005) *A combinatorial view of graph Laplacians*. Technical report, Max Planck Institute for Biological Cybernetics.

[3] von Luxburg, U. (2007) *A Tutorial on Spectral Clustering*. CoRR abs/0711.0189.

[4] Aldrich, J. (2006) *Eigenvalue, eigenfunction, eigenvector, and related terms* , in Jeff Miller (Editor), Earliest Known Uses of Some of the Words of Mathematics.

[5] Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., & Van der Vorst, H. (2000) *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM Press, Philadelphia, PA.

[6] Lanczos, C. (1950) *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bureau Standards, Sec. B, 45, pp. 255282.

[7] Lloyd, S. P. (1982) *Least squares quantization in PCM*, IEEE Transactions on Information Theory 28 (2): 129137.

[8] Cullum, J. & Willoughby, R. (1985) *Lanczos algorithms for Large Symmetric Eigenvalue Computations*, Birkhauser.

[9] Song, Y., Chen, W., Bai, H., Lin, C., Chang, E.Y. (2008) *Parallel Spectral Clustering*. ECML/PKDD (2): 374-389.

[10] Charikar, M. (2000) *Greedy approximation algorithms for finding dense components in a graph*. APPROX 2000:84-95.

[11] Bahmani, B., Kumar, R., Vassilvitskii, S. (2012) *Densest Subgraph in Streaming and MapReduce*. PVLDB 5(5):454-465.

[12] Gibson, D. Kumar, R., Tomkins, A. (2005) *Discovering Large Dense Subgraphs in Massive Graphs*. VLDB:721-732.

[13] Miao, G., Song, Y. Zhang, D., Bai, H. (2008) *Parallel Spectral Clustering Algorithm for Large-Scale Community Data Mining*, WWW2008.

[14] Fowlkes, C., Belongie, S., Chung, F. & Malik, J. (2004) *Spectral grouping using the Nystrom method*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(2):214225.

[15] Dean, J., Ghemawat, S. (2004) *MapReduce: Simplified Data Processing on Large Clusters*. OSDI:137-150.

[16] Hefeeda, M., Gao, F., Abd-Almageed, W. (2012) *Distributed approximate spectral clustering for large-scale datasets*. HPDC:223-234.

[17] Cordeiro, R.L.F., Traina Jr.,C., Traina, A.J.M. López, J., Kang, U., Faloutsos, C. (2011) *Clustering very large multi-dimensional datasets with MapReduce*. KDD:690-698.

[18] Maschhoff, K.J., Sorensen, D.C. (1996) *PARPACK: An Efficient Portable Large Scale Eigenvalue Package for Distributed Memory Parallel Architectures*. PARA: 478-486.

[19] Wu, K., Simon,H. (1999) *A parallel Lanczos method for symmetric generalized eigenvalue problems*. Computing and Visualization in Science, 2:37-46.

[20] Hernández, V., Román, J.E., Vidal, V. (2005) *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*. ACM Trans. Math. Softw. (TOMS) 31(3):351-362.

[21] Kang, U., Meeder, B., Faloutsos, C. (2011) *Spectral Analysis for Billion-Scale Graphs: Discoveries and Implementation*. PAKDD:13-25.

[22] Kang, U., Chau, D.H., Faloutsos, C. (2012) *Pegasus: Mining billion-scale graphs in the cloud*. ICASSP:5341-5344.

[23] Kang, U., Tsourakakis, C.E., Faloutsos, C. (2009) *PEGASUS: A Peta-Scale Graph Mining System*. ICDM:229-238.

[24] Zhao, W., Ma, H., He,Q. (2009) *Parallel K-Means Clustering Based on MapReduce*. CloudCom:674-679.

[25] Arthur, D., Vassilvitskii, S. (2007) *k-means++: the advantages of careful seeding*. SODA:1027-1035.

[26] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S. (2012) *Scalable K-Means++*. PVLDB 5(7):622-633.

[27] Lancichinetti, A., Fortunato, S. & Radicchi, F. (2008) *Benchmark graphs for testing community detection algorithms*. Phys. Rev. E 78 (4): 046110.