Large Graph Clustering Using DCT-Based Graph Clustering

Nikolaos Tsapanos, Anastasios Tefas, Nikolaos Nikolaidis and Ioannis Pitas
Department of Informatics
Aristotle University of Thessaloniki
Box 451, Thessaloniki, GR 54124
Email: niktsap@aiia.csd.auth.gr, tefas@aiia.csd.auth.gr, nikolaid@aiia.csd.auth.gr, pitas@aiia.csd.auth.gr

***Abstract*—With the proliferation of the World Wide Web, graph structures have arisen on social network/media sites. Such graphs usually number several million nodes, i.e., they can be characterized as Big Data. Graph clustering is an important analysis tool for other graph related tasks, such as compression, community discovery and recommendation systems, to name a few. We propose a novel extension to a graph clustering algorithm, that attempts to cluster a graph, through the optimization of selected terms of the graph weight/adjacency matrix Discrete Cosine Transform.**

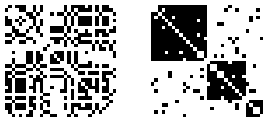
I. Introduction

Graph clustering, i.e., grouping graph nodes that are closely interconnected has an important role in graph analysis applications. In this paper, we will present a genetic algorithm for graph clustering that is designed to be fast and whose memory requirements are linear with respect to the number of both graph nodes and graph edges.

There has been an abundance of works that deal with the clustering of graph nodes. For an in-depth survey on graph clustering, the interested reader can refer to [1]. One of the more theoretically elegant types of approaches is using the *graph spectrum* [2] to perform the clustering. Such methods are referred to as *spectral graph clustering* methods, a popular example of which is the *Normalized Cut*, or *NCUT*, algorithm [3].

We base the method proposed in this paper on a novel approach to graph node clustering presented in [4], which uses optimization on selected terms of the *Discrete Cosine Transform* (*DCT*) [5] of the graph adjacency matrix to perform the clustering. We will attempt to solve the parameterization issues that were observed in that work, by a) constructing a padded graph based on the input graph, b) defining a different DCT-related objective function to optimize, c) restricting the way optimization can be performed and d) using the new method to recursively split the clusters into subclusters.

The paper is organized as follows: Section II briefly describes the initial idea presented in [4], section III details the modifications that are proposed on a theoretical level, while section IV describes how the computations can be significantly sped up. Section V provides the results of the comparative evaluation experiments and Section VI concludes the paper.



(a) (b)

Fig. 1. a) The adjacency matrix of a graph with 3 clusters when the node order is random (edges are marked in black), b) The adjacency matrix of the same graph, when the nodes are placed in consecutive rows/columns for each cluster.

II. DCT-based Graph Clustering

As proposed in [4], the main idea behind using the DCT of the graph adjacency matrix to perform clustering on a graph, stems from a simple observation. When the nodes of each cluster appear in consecutive places in the order used to construct the adjacency matrix, then large blocks of edges appear along the image diagonal. This is illustrated in Figure 1.

In the DCT domain, large, square blocks along the diagonal correspond to large values in the diagonal terms of the DCT, as can be seen in Figure 2. Swapping the order of nodes in the graph adjacency matrix causes the DCT term values to fluctuate. It is, therefore, possible to bring nodes of the same cluster together in the adjacency matrix by maximizing the sum of the first k terms of the DCT diagonal. This can be achieved by using *Simulated Annealing* (*SA*) [4].

The issue that arises, however, is the determination of the parameter k. In general, k should be close to the number of clusters. As Figure 3 shows, using the wrong values for this parameter can cause problems. In this example, there are 3 clusters in the input graph. If the number of clusters is underestimated, then the two smaller clusters are merged together (Figure 3b). If the number of clusters is overestimated, then the biggest cluster is split (Figure 3d). This can be useful, if we want the graph to be clustered into a specific number of clusters. However, in the general case, we must correctly guess the number of clusters, otherwise the method will underperform. Furthermore, depending on the size difference between the smallest and biggest cluster, it may be impossible to choose a value for k that will not split the biggest cluster or merge the smaller one into another cluster. It is this issue that we address in this paper.

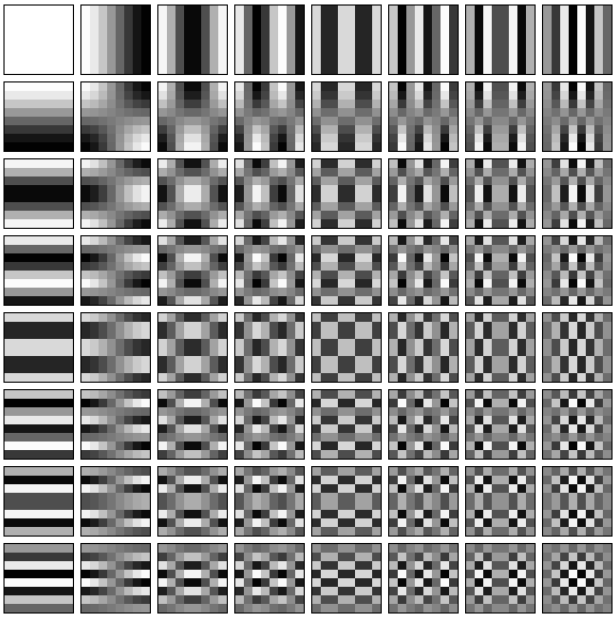


Fig. 2. The basis functions of an 8×8 matrix.

Fig. 3. a) Input graph, b) number of clusters underestimated ($k = 2$), c) number of clusters correctly estimated ($k = 3$), d) number of clusters overestimated (k=5).

III. Divisive DCT-based Graph Clustering

This section details our proposed modifications to [4] that aim to de-parameterize the method and improve clustering performance. The optimization method remains Simulated Annealing.

We will first describe the changes to the input graph adjacency matrix of N nodes. We pad the matrix with dummy nodes to obtain a *padded graph* with adjacency matrix $\mathbf{B}$ as follows:

- Group 1: nodes in matrix places 1-N are dummy nodes, not connected to any other node.
- Group 2: nodes in matrix places $(N+1)$-$(3N/2)$ are auxiliary nodes, connected to every node in groups 2, 3 and 4 with weight 0.5.
- Group 3: nodes in matrix places $(3N/2+1)$-$(5N/2)$ are the real nodes of the input graph, connected in the same way with each other and also groups 2 and 4 (weight 0.5).
- Group 4: nodes in matrix places $(5N/2+1)$-$3N$ are auxiliary nodes, connected to every node in groups 2, 3 and 4 with weight 0.5.
- Group 5: nodes in matrix places $(3N+1)$-$8N$ are dummy nodes, not connected to any other node.

This creates an $8N \times 8N$ matrix, as illustrated in Figure 4.

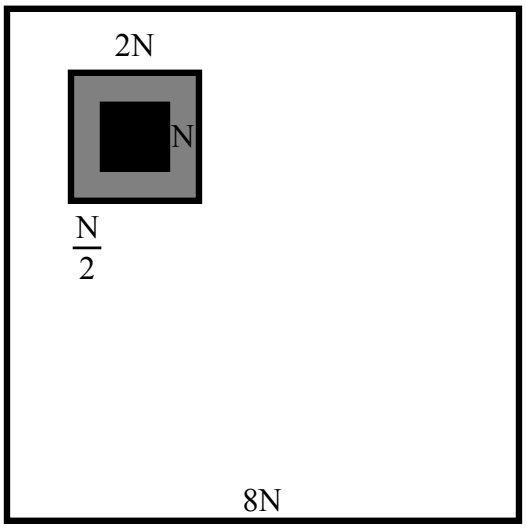


Fig. 4. The padded graph. The black square is the original graph (node group 3 with input graph weights), surrounding it is the gray square (node groups 2 and 4 with weights 0.5), while surrounding everything is the white square (node groups 1 and 5 with weights 0).

Now we will describe the restrictions we place on node swapping. Instead of allowing any two nodes to swap places in the adjacency weight matrix, we attempt to optimize the objective function $f(\mathbf{B})$ by swapping nodes in the following way:

- Select a random node with position i in the range of N-$3N$.
- Swap the selected node with the node in position $5N+i$.
- Calculate the change in the objective function $f(\mathbf{B})$.
- Accept the swap (transition) with the appropriate probability, as per SA operation.

In order to visualize what happens with these swap, please refer to Figure 5. When the optimization starts, all the non-zero weights are concentrated in the square marked C_1. When the first node is swapped from C_1, some of its weights move to squares C_2, I_1 and I_2. As the optimization continues weights move between these squares. Note that squares I_1 and I_2 contain the weights of the edges connecting the nodes in C_1 with the nodes in C_2. Suppose that the graph has 2 clusters. It is easy to see that, if every node of one cluster was in C_1 and every node of the other cluster was in C_2, then the weights in C_1 and C_2 would be very strong, as they correspond to internal edges between nodes of the same cluster. Respectively, weights in I_1 and I_2 would be fairly weaker, as they correspond to inter-cluster edges between the 2 clusters. If there are more than 2 clusters in the graph, it suffices that I_1 and I_2 only contain inter-cluster edges, since we can recursively cluster the nodes of C_1 and C_2 in the same way, until a stopping criterion is met.

The task, at this point, is to devise an objective function that a) minimizes the inter-cluster weights in I_1 and I_2 and

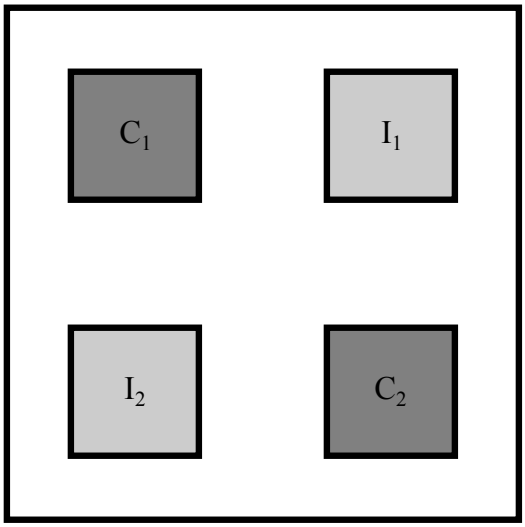


Fig. 5. All the possible positions in which the non-zero weights can be found, under the swap restriction we placed. Ideally, when the optimization ends, all the nodes of one cluster will end up in C_1 and all the nodes of the other cluster in C_2, with the inter-cluster edge weights residing in I_1 and I_2

b) splits the graph somewhat evenly, as we don't want C_2 to be empty. Let $DCT_{\mathbf{B}}(x, y)$ refer to the (x, y) term of the DCT of matrix $\mathbf{B}$. Looking back at Figure 2, we can see that $DCT_{\mathbf{B}}(4, 2)$ and $DCT_{\mathbf{B}}(2, 4)$ have high values, when there are weights in I_1 and I_2. As it follows, we want to minimize $q_1 = DCT_{\mathbf{B}}(4, 2) + DCT_{\mathbf{B}}(2, 4)$. The problem is that q_1 is already at its minimum value, when every node is in C_1. We, therefore, need to force the objective function to move at least some nodes to C_2. Note that the DCT terms $DCT_{\mathbf{B}}(1, 2)$ and $DCT_{\mathbf{B}}(2, 1)$ have high values, when the nodes are concentrated in C_1 and low values, when the nodes are concentrated in C_2. We must also minimize $q_2 = |DCT_{\mathbf{B}}(1, 2)| + |DCT_{\mathbf{B}}(2, 1)|$. One last problem stems from the conflict between these two goals, as reducing one increases the other. Focusing on minimizing q_1 will not split the graph. Focusing on minimizing q_2 may split clusters, as it will force the graph to be split exactly evenly. Our solution is to employ an exponential function $e^{-\alpha q_2}$ so that, when q_2 is high, $e^{-\alpha q_2} >> q1$. This forces the optimization to move at least some nodes in C_2. Additionally, once q_2 reaches a fraction of its initial value (dictated by α), then $e^{-\alpha q_2}$ becomes almost a constant and will not force exactly even graph splits. The objective function for the optimization is, therefore, defined to be:

$$f(\mathbf{B}) = q_1 + Ne^{-\alpha q_2}.$$

When the minimization is over, the input graph has been partitioned into 2 subgraphs. We can apply the same method again to each subgraphs and keep splitting the resulting graphs, until a subgraph is considered too dense to be further split.

IV. Fast implementation

In our explanation of the algorithm, we used the DCT of a padded graph matrix. Using such a matrix in practice requires $O(n^2)$ memory and such an implementation would be infeasible. Additionally, recomputing the DCT is extremely inefficient. It is, in fact, possible to perform all the required computations through a node adjacency list. This is even more useful, when the graph is rather sparse.

Let us consider $DCT_{\mathbf{B}}(4, 2)$, as all the other terms are similar with this one. Note that

$$DCT_{\mathbf{B}}(4, 2) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} \cos(\frac{\pi}{n}(j + \frac{1}{2})2) \cos(\frac{\pi}{n}(i + \frac{1}{2})4).$$

The DCT is computed once at the start of the process. We maintain the current position of each node in a global vector and its adjacency list in its own adjacency vectors. The contribution of graph node k that is currently in position i to $DCT_{\mathbf{B}}(4, 2)$ is

$$\sum_{j=1}^{n} b_{ij} \cos(\frac{\pi}{n}(j + \frac{1}{2})2) \cos(\frac{\pi}{n}(i + \frac{1}{2})4).$$

Moving a node from one group to the other essentially changes the position of the node from i to i'. We can calculate the current contribution, by going through its adjacency list vector and retrieving the current positions of its neighbors from the global vector, thus determining the value for b_{ij}. Nodes that are not connected to node k yield $b_{ij} = 0$ and are not considered in the calculation.

After computing the contribution, we can subtract it from $DCT_{\mathbf{B}}(4, 2)$. We can then move the node by updating its position in the global position vector, calculate the node's contribution in the new position and add it back into $DCT_{\mathbf{B}}(4, 2)$. In this way, the changes in the objective function can be calculated in a minimal way.

In this implementation, if N is the bumber of nodes and M is the number of edges, then the memory requirements for each node i are storing its adjacency lists, which has a size of d_i, where d_i is the node's degree, plus its place in the padded matrix. Note that $\sum_i d_i = M$ and, therefore, the total memory requirements are $O(M + N)$.

V. Experiments

In order to compare our novel clustering method with NCUT, we ran our experiments on the same data as the facial image clustering method, presented in [6], which uses NCUT. The facial images were extracted from the movies included in the database described in [7]. The weight matrix $\mathbf{A}$ was constructed using a Normalized Mutual Information measure, as described in [8]. The matrix was then normalized as follows: $\frac{1}{\max(\mathbf{A})}(\mathbf{A} - \min(\mathbf{A}))$. The clustering was achieved through the optimization process detailed in Section III. The criterion for stopping the recursive cluster splitting process was checking whether $\max(\mathbf{A}) - \min(\mathbf{A}) \leq 0.7$. Results for the proposed method and the method in [6] are shown in Table I. Overviewing the results, we conclude that the performance of the proposed method is comparable to that of [6].

In order to evaluate our approach on a large graph, we selected the youtube dataset from the Stanford Network Analysis Project [9]. The dataset is provided with community ground truth. We selected to only include communities, which have more than 100 members. This resulted in 105 communities.

The extracted graph of users that belong to these communities contains 17126 nodes and 212136 edges. The ground truth labels for each node were determined, according the largest community the node belonged to. The comparison between the labels provided by the proposed approach and the ground truth was carried out using the F-measure [10]. The runtime was 33 seconds and the recorded F-measure was 0.1482, which is consistent with the NMI performance, another measure closely related to F-measure, of other state of the art clustering algorithms on Big Data graph clustering [11].

VI. Conclusions

In this paper we have presented a novel improvement on an original approach to graph clustering and used it to perform facial image clustering and youtube community clustering. By properly constructing a padded graph and only allowing very specific node swaps we were able to define an objective function, whose minimization leads to splitting the graphs in such a way that clusters remain mostly together. Experimental results indicate that this approach performs very comparably with the spectral graph clustering approach in [6]. Also note that the approach in [6] requires providing different values for 2 parameters in order to perform this well, while the method presented here does not require changing values for any parameters. Additionally, we used our approach to perform fast and efficient clustering on a large graph.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 316564 (IMPART). This publication reflects only the authors views. The European Union is not liable for any use that may be made of the information contained therein.

References

[1] Satu Elisa Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[2] Fan R. K. Chung, *Spectral Graph Theory*.

[3] Jianbo Shi and Jitendra Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888–905, 1997.

[4] N. Tsapanos, I. Pitas, and N. Nikolaidis, "Graph representations using adjacency matrix transforms for clustering," in *Electrotechnical Conference (MELECON), 2012 16th IEEE Mediterranean*, March 2012, pp. 383 –386.

TABLE I. Comparative evaluation between [6] and the proposed method.

Movie	[6]	Proposed
Bringing out the dead	87.31%	100%
Erin Brockovich	100%	96.49%
Forest Gump	98.63%	92.85%
Gandhi	97.62%	93.13%
I am Sam	93.69%	93.69%
Indiana Jones and the last crusade	90.21%	99.02%
Kids	90.56%	93.64%
Lord of the rings	96.00%	96.00%
Mission to Mars	87.69%	81.27%
The pianist	97.20%	98.83%
Pulp fiction	98.52%	100%
The Godfather	100%	98.29%

[5] I. Pitas, *Digital Image Processing Algorithms and Applications*, A Wiley-Interscience publication. Wiley, 2000.

[6] N.Vretos C.Chrysouli and I.Pitas, "Face clustering in videos based on spectral clustering techniques," in *Asian Conference on Pattern Recognition (ACPR2011)*, November 2011.

[7] Ivan Laptev, Marcin Marszalek, Cordelia Schmid, Benjamin Rozenfeld, Inria Rennes, Irisa Inria Grenoble, and Lear Ljk, "Learning realistic human actions from movies," in *In: CVPR. (2008*, 2008.

[8] N. Vretos, V. Solachidis, and I. Pitas, "A mutual information based face clustering algorithm for movie content analysis," *Image Vision Comput.*, vol. 29, pp. 693–705, Sept. 2011.

[9] Jaewon Yang and Jure Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, New York, NY, USA, 2012, MDS '12, pp. 3:1–3:8, ACM.

[10] David M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation," Tech. Rep. SIE-07-001, School of Informatics and Engineering, Flinders University, Adelaide, Australia, 2007.

[11] Xiao Cai, Feiping Nie, and Heng Huang, "Multi-view k-means clustering on big data.," in *IJCAI*, Francesca Rossi, Ed. 2013, IJCAI/AAAI.