IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 8, August 2016

Graph Clustering: Modified BFS Algorithm

Ishwar Baidari[1], Ajith Hanagwadimath[2]

Associate Professor, Department of Computer Science, Karnatak University, Dharwad, India[1]

Research Scholar, Dept of Computer Science, Karnatak University, Dharwad, India[2]

Abstract: Graphs are structures formed by a set of vertices also called nodes and set of edges that are connections between pairs of vertices. Graph clustering is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters. Here we present a polynomial time algorithm clustering a given graph according to modified BFS algorithm.

Keywords: Clustering, Vertices, nodes, BFS.

I. INTRODUCTION

A. Graph Clustering

1) Models for clustered graphs:

Gilbert [1] presented in 1959 a process of generating uniform random graphs with n vertices: each of the $\binom{n}{2}$ possible edges is included in the graph with probability p, considering each pair of vertices independently. In such uniform random graphs, the degree distribution is Poissonian. Also, the presence of dense clusters is unlikely as the edges are distributed by construction uniformly, and hence no dense clusters can be expected.

A generalization of the Gilbert model, especially designed to produce clusters, is the planted l-partition model [2]: a graph is generated with n= l.k vertices that are partitioned into l groups each with k vertices. Two probability parameters p and q <p are used to construct the edge set: each pair of vertices that are in the same group share an edge with the higher probability p, whereas each pair of vertices in different groups shares an edge with the lower probability r. The goal of the planted partition problem is to recover such a planted partition into l clusters of k vertices each, instead of optimizing some measure on the partition. McSherry [3] discusses also planted versions of other problems such as k-clique and graph colouring.

2) Cluster Properties:

No single definition of a cluster in graphs is universally accepted, and the variants used the literature are numerous [4]. In the setting of graphs, each cluster should intuitively be connected: there should be at least one, preferably several paths connecting each pair of vertices within a cluster. If a vertex u cannot be reached from a vertex v, they should not be grouped in the same cluster. Furthermore, the paths should not be internal to the cluster. As a consequence, when clustering a disconnected graph with known components, the clustering should usually be conducted on each component separately, unless some global restriction on the resulting clusters is imposed. In some applications one may wish to obtain clusters of similar order and /or density, in which case the clusters computed in one component also influence the clustering's of other components. It is generally agreed upon that a subset of vertices forms a good cluster if the induced subgraph is dense, but there are relatively few connections from the included vertices to vertices in the rest of the graph [5, 6, 7, 8, 9].

One measure that helps to evaluate the sparsity of connections from the cluster to the rest of the graph is the cut size c (C, V/C). The smaller the cut size, the better "isolated" the cluster. Determining when a cluster is dense is naturally achieved by computing the graph density. We refer to the density of the subgraph induced by the cluster as the internal or intra-cluster density:

$$\delta_{int}(C) = \frac{|\{\{v,u\} \mid v \in C, u \in C\}|}{|C|(|C|-1)}$$

The intercluster density of a given clustering of a graph G into k clusters C1, C2……Ck is the average of the intercluster densities of the included clusters:

$$\delta_{int}(G|C_1 \ldots\ldots, C_k) = \frac{1}{k}\sum_{i=1}^{k} \delta_{int}(C_i)$$

The external or inter-cluster density is of a clustering is defined as the ratio of intercluster edges to the maximum number of intercluster edges possible, which is effectively the sum of the cut sizes of all the clusters, normalized to the range [0,1]:

$$\delta_{ext}(G|C_1 \ldots\ldots, C_k) = \frac{|\{\{v,u\} \mid v \in C_i, u \in C_j, i \neq j\}|}{n(n-1) - \sum_{i=1}^{k}(|C_i|(|C_i|-1))}$$

Globally speaking, the internal density of a good clustering should be notably higher than the density of the graph $\delta(G)$(Eq. (3)) and the inter cluster density of the clustering should be lower than the graph density [10].

For general clustering tasks, fuzzy clustering algorithms have been proposed [11, 12], as well as validity measures [13]. Within graph clustering, not much work can be found on fuzzy clustering, and in general, the past decade has been quiet on the area of fuzzy clustering. Yan and Hsiao

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 8, August 2016

[14] present a fuzzy graph clustering algorithm and apply it to circuit partitioning. A study on general clustering methods using fuzzy set theory is presented by Dave and Krishnapuram [15].
A fuzzy graph GR = (V,R) is composed of a set of vertices and a fuzzy edge-relation R that is reflexive and symmetrical together with a membership function μR assigns to each fuzzy edge a level of "presence" in the graph [16]. Different nonfuzzy graphs can be obtained by thresholding $\mu R(v, u) \geq \tau$ are included as edges in $G\tau$. The graph $G\tau$ is called a cut graph of GR.
Dong et al. [16] present a clustering method based on a connectivity property of fuzzy graphs assuming that the vertices represent a set of objects that is being clustered based on a distance measure. Their algorithm first preclusters the data into subclusters based on the distance measure, after which fuzzy graph if constructed for each subcluster and a cut graph of the resulting graph is used to define what constitutes a cluster. Dong et al. also discuss the modifications needed in the current clustering upon updates in the database that contains the objects to be clustered.
Fuzzy clustering has not been established as a widely accepted approach for graph clustering, but it offers more relaxed alternative for applications where assigning each vertex to just one cluster seems restricting while the vertex does relate more strongly to some of the candidate clusters than to others.

B. Clusters for different classes of graphs

It is common that in applications, the graphs are not just simple, unweighted and undirected. If more than one edge is allowed between two vertices, instead of a binary adjacency matrix it is customary to use a matrix that determines for each pair of vertices how may edges they share. Graphs with such edge multiplicities are called multigraphs.
Also, should the graph be weighted, cutting an important edge (with a large weight) when separating a cluster is to be punished more heavily than cutting a few unimportant edges (with very small weights). Edge multiplicities can in essence be treated as edge weights, but the situation naturally gets more complicated if the multiple edges themselves have weights.

Luckily, many measures extend rather fluently to incorporate weights or multiplicities. It is especially easy when the possible values are confined to a known range, as this range can be transformed into the interval [0, 1] where one corresponds to as "full" edge, intermediate values to "partial" edges, and zero to there being no edge between two vertices. With such a transformation, we may compute density not by counting edges but summing over the edge weights in the unit line: the internal density of a cluster e (Eq. (22)) on Section 3.2 is rewritten as to account for the degree of "presence" of the edges.

$$\delta_{int}(C) = \frac{1}{|C|(|C|-1)} \sum_{\substack{\{v,u\} \in E \\ v \in C, u \in C}} \omega(v,u)$$

Now a cluster of high density has either many edges or important edges, and a low-density cluster has either few or unimportant edges. It may be desirable to do a nonlinear transformation from the original weight set to the unit line to adjust the distribution of edge importance if the clustering results obtained by the linear transformation appear noisy or otherwise unsatisfactory.

C. Bipartite graphs

A bipartite graph is a graph where the vertex set V can be split in two sets A and B such that all edges lie between those two sets: if(v, w) ∈ E, either v ∈ A and w ∈ B or v ∈ B and w ∈ A. Such graphs are natural for many application areas where the vertices represent two distinct classes of objects, such as customers and products; an edge could signify for example that certain customer has bought a certain product. Possible clustering tasks could be grouping the customers by the types of products they purchase or grouping products purchased by the same people – the motivation could be targeted marketing, for instance. Carrasco et al. [17] study a graph of advertisers and keywords used in advertisements to identify submarkets by clustering.

D. Directed graphs

Up to now, we have been dealing with undirected graphs. Let us turn into directed graphs, which require special attention, as the connections that are used in defining the clustering are asymmetrical and so is the adjacency matrix. This causes relevant changes in the structure of the Laplacian matrix and hence makes spectral analysis more complex.

Web graphs [18] are directed graphs formed by web pages as vertices and hyperlinks as edges. A clustering of a higher-level web graph formed by all Chilean domains was presented by Virtanen [19]. Clustering of web pages can help identify topics and group similar pages. This opens applications in search-engine technology; building artificial clusters is known to be a popular trick among websites of adult content to try to fool the PageRank algorithm [20] used by Google to rate the quality of websites.

E. Vertex similarity

There are many clustering algorithms based on similarities between the vertices. Should the vertices represent documents, for example, one could compute content-based similarity matrix as a basis for the clustering, attempting to group together vertices that are not only well connected but also similar to each other. The higher the similarity, the stronger the need to cluster the vertices together. Computing such similarities is not necessarily simple, and in some cases evaluating the similarity of two vertices may turn out to be a task even more complex than the clustering of the graph once the similarities are known.
If a similarity measure has been defined for the vertices, the cluster should contain vertices with close-by values and exclude those for which the values differ significantly from the values of the included vertices. If instead of

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 8, August 2016

similarity, we use a distance measure, the cluster boundary should be located in an area where including more of the outside vertices would drastically increase the intracluster distances (for example, the sum of squares of all-pairs distances). Hence, with distance measures, it is desirable to cluster together vertices that have small distances to each other.

F. Complexity of global clustering

In this section we discuss some related problems where a dataset- which can be represented as a (weighted) complete graph is divided into clusters that optimize a certain criteria. Understanding of the approximability and the algorithms for these problems helps to understand how good global clustering algorithms can be.

The minimum k-clustering problem is the combinatorial optimization problem where a finite data set D is given together with a distance function d: D x D → N, where d satisfies the triangle inequality (Eq. (28)). The task is to partition D into k clusters C1, C2……Ck, where Ci ∩ Cj = ∅ for i ≠ j, such that the maximum intercluster distance is minimized (i.e. the maximum distance between two points assigned to the same cluster). This problem is approximable within a factor of two, but not approximable within (2 - ∈) for any ∈ > 0 [21, 22].

II. APPLICATIONS OF GRAPH CLUSTERING

As has been emphasized repeatedly throughout the survey, the task of clustering is highly application-specific. In this section we review some of the key application areas of graph clustering, although it is not to be forgotten that many problems allow the utilization of other representations as well and hence clustering algorithms for feature vectors or others kinds of classification systems, for example, may equally be applied. We begin by viewing how data sets composed of points in n-dimensional space can be transformed into graphs.

G. Data transformations

The range of interesting clustering applications is wide, as many if not practically all systems of interacting (or simply coexisting) entities can be modelled in some way as graphs. For data that are not readily in graph, several transformations into graph representations are possible. In this section we discuss some of the various possibilities to convert feature-vector data into graph format. Transformations vice versa exist as well [23], but as the focus of this survey are graph-theoretical clustering algorithms, we do not address those.

One option on how to convert feature-vector data into graph format is the Delaunay graph. The Delaunay graph of a set of points on a plane can be constructed by representing each point by a vertex and placing an edge between each pair of points that are Voronoi neighbours [24]. The approach naturally generalizes to higher dimensions. Two points are Voronoi neighbours if their Voronoi cells are adjacent [25]. A Voronoi cell of a datum is formed by those points in the data space that are closer to that data point than any other. The boundaries of the Voronoi cells are hyperplanes that partition the space in which the data lie.

III.ALGORITHM

```
Cluster(G)
1       for each vertex u ∈ G.V
2       {
3           u.color = White
4           u.cluster = Nil
5       }
6       for each edge (u,v) ∈ G.E
7       {
8           (u,v).color = Green
9           SetLabel ( (u,v), Undetermined)
10      }
11      Cluster_Count = 0
12      Cluster_Limit = 3
13      Q = ∅
14      for each vertex u ∈ G.V
15      {
16              if( u.color == white)
17              {
18                      Cluster_Count = Cluster_Count+1
19                      Enqueue (Q,u)
20                      Set Cluster_Cluster_Count = ∅
21                      count = 0
22                      while (Q ≠ ∅)
23                      {
24                          u = Dequeue(Q)
25                          for each vertex v ∈ G.Adj[u]
26                          {
27                              if (v.color == White)
28                              {
29                                  v.color = Gray
30                                  Enqueue (Q,v)
31                              }
32                          }
33                          u.color = Black
34                          u.cluster = Cluster_Count
35                          Set Cluster_Cluster_Count = Set Cluster_Cluster_Count ∪{u}
36                          count = count + 1
37                          if (count == Cluster_Limit)
38                          {
39                              if (Q ≠ ∅)
40                              {
41                                  Cluster_Count = Cluster_Count+1
42                                  Set Cluster_Cluster_Count = ∅
```

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 8, August 2016

```
43                                          u          =
Dequeue(Q)
44                                          while      (Q
≠∅)
45                                                     {
46                                    v = Dequeue(Q)
47                                     v.color = White
48                                                     }
49                                        Enqueue (Q,u)
50                                         count = 0
51                                         }
52                                  }
53
54                          }End of While

55                  }End of If
56          }End of for
57          for each edge (u,v) ∈ G.E
58          {
59                  if ( u.cluster == v.cluster)
60                    {
61                          SetLabel ((u,v), Cluster_Edge)
62                    }
63                  else
64                    {
65                    (u,v).color = Red
66                     SetLabel((u,v),
Cluster_Connecting_Edge)
67                    }
68          }
```

A. Some of the attributes for vertices and edges assumed in this algorithm are as follows

- For a vertex u ∈ G.V the attribute u.color holds the color of the vertex u.
- For a vertex u ∈ G.V the attribute u.cluster holds the number of the cluster in which the vertex u is present. For ex: if the vertex 'u' is present in cluster 1 then the u.cluster attribute will be having the value u.1.
- For a edge (u,v) ∈ G.E the attribute (u,v).color holds the color of the edge (u,v).
- For a a edge (u,v) ∈ G.E the SetLabel function assigns a label depending upon the situation such as,
  - If the edge connects two vertices present in the same cluster then the label will be set as 'Cluster_Edge'.
  - If the edge connects two vertices present in different clusters then the label will be set as 'Cluster_Connecting_Edge'.

B. Working of the Algorithm : The working of the above algorithm is as explained below

- The for loop in lines 1-5 initializes all the vertices of the given graph. The u.color attribute is set to 'White' for all the vertices in line 3. And the u.cluster attribute is set to 'Nil' for all the vertices in line 4.
- The for loop in lines 6-10 initializes all the edges of the given graph. The (u,v).color attribute of all the edges is set to 'Green' in line 8 and the label of all the edges is set to 'Undetermined' in line 9.
- In line 11 a variable 'Cluster_Count' is declared and initialized with zero. This variable gives the total number of clusters formed.
- In line 12 a constant 'Cluster_Limit' is declared and initialized with value 3. This value gives the upper limit on the size of a cluster.
- A queue 'Q' is declared and initialized with null in line 13.
- The for loop in lines 14-56 scans every vertex u ∈ G.V of the given graph and forms clusters accordingly.
  - The condition in line 16 checks whether the color of vertex u==white, if so, then the variable 'Cluster_Count' is incremented by 1 in line 18. The vertex 'u' is enqueued into the queue 'Q' in line 19. A set data structure 'Cluster_Cluster_Count' is declared and initialized with null in line 20. And a variable 'count' is declared and initialize with 0 in line 21.
  - The While loop in lines 22-44 forms the clusters according to the advance of the BFS algorithm (limiting the number of nodes in a cluster to 3).

The vertex 'u' is dequeued from the front of the queue 'Q' in line 24. The for loop in lines 25 to 32 scans the adjacency list of the vertex 'u' which is dequeued in line 24. The condition in line 27 checks whether the color of the vertex 'v' is white, if so, then the v.color attribute is set to Gray in line 29 and the vertex 'v' is enqueued into the rear end of the queue 'Q' in line 30. The color attribute of vertex 'u' is set to 'Black' in line 33. The 'u.cluster' attribute is assigned the value Cluster_Count in line 34. The vertex 'u' is added to the set 'Cluster_Cluster_Count' in line 35 and the value of the variable 'count' is incremented by 1 in line 36.

- The condition in line 37 checks whether the value of the variable 'count' is Equal to 'Cluster_Limit'. If so, then the condition in line 39 checks whether the queue 'Q' is not empty, if so then, the variable 'Cluster_Count' is incremented by 1 in line 41. A set data structure 'Cluster_Cluster_Count' is declared and initialized with null in line 42. A vertex 'u' which is at the rear end of the queue 'Q' is dequeued in line 43 (which will be later used as the starting point for new cluster). And the while loop in lines 44-48 dequeues every vertex 'v' from the queue 'Q' and sets the v.color attribute to white in line 47. The vertex 'u' which was dequeued in line 43 will be enqueued into the queue 'Q' in line 49 and the variable 'count' is once again set to zero (0) in line 50.
- The for loop in lines 57-68 is used to determine the edges as either 'Cluster_Edges' or 'Cluster_Connecting_Edges'
  - The condition in line 59 checks whether the attributes v.cluster == u.cluster , if so, then both the vertices belong to the same cluster and the label 'Cluster_Edge' is set to the edge (u,v) in line 61.
  - If the condition in line 47 fails then, the color attribute of the edge (u,v) is set to Red in line 65 and the Label 'Cluster_Connecting_Edge' is set to the edge (u,v) in line 66.

Copyright to IARJSET
DOI 10.17148/IARJSET.2016.3844

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 3, Issue 8, August 2016

IV. TIME COMPLEXITY

- The for loop in lines 1-5 initializes every vertex u ∈ G.V in the graph and hence it takes O (V) time.
- The for loop of lines 6-10 initializes every edge (u,v) ∈ G.E in the graph and hence it takes O(E) time.
- The for loop in line 14-56 executes once for every vertex u ∈ G.V in the graph and the logic used to form clusters within this for loop is same as that of BFS algorithm. Hence the total time taken by this section of the algorithm will be equal to O(V+E) time.
- The for loop in lines 57-68 scans every edge (u,v) ∈ G.E in the graph and hence it takes O(E) time.

Hence the total time complexity of the above algorithm is;

$$O(V) + O(E) + O(V+E) + O(E) = O(V + E)$$

REFERENCES

[1] E. N. Gilbert, Random graphs, Annals of Mathematical Statistics 30 (4) (1959) 1141-1144.
[2] Condon, R.M. Karp, Algorithms for graph partitioning on the planted partition model, Random Structures & Algorithms 18 (2) (2001) 116-140.
[3] F. McSherry, Spectral partitioning of random graphs, in: Proceedings of the Fourty-Second IEEE Symposium on Foundations of Computer Science, FOCS, IEEE Computer Society Press, Washington, DC, USA, 2001.
[4] J. Edachery, A. Sen, F.J. Brandenburg, Graph clustering using distance-k cliques, in: Proceedings of the Seventh International Symposium on Graph Drawing, in: Lecture Notes in Computer Science, vol. 1731, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1999.
[5] L.M.A. Bettencourt, Tipping the balance of a small world, Tech. Rep. MIT-CTP-3361 (cond-mat/0304321 at arXiv.org), Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA, USA, 2002.
[6] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences, USA 99 (2002) 8271-8276.
[7] R. Kannan, S. Vempala, A. Vetta, On clusterings- good, bad and spectral, Journal of the ACM 51 (3) (2004) 497-515.
[8] J. M. Kleinberg, S. Lawrence, The structure of the Web, Science 294 (5548) (2001) 1849-1850.
[9] M.E.J. Newman, Fast algorithm for detecting community structure in networks, Physical Review E 69 (6) (2004) 066133.
[10] M.E.J. Newman, Detecting community structure in networks, The European Physical Journal B 38 (2) (2004) 321-330.
[11] F. Hoppner, F. Kalwonn, R. Kruse, T. Runkler, Fuzzy Cluster Analysis: Methods for classification, Data Analysis and Image Recognition, John Wiley & Sons, Inc., Hoboken, NJ, USA, 1999.
[12] Gath, A.B. Geva, Unsupervised optimal fuzzy clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (7) (1989) 773-780.
[13] [239]
[14] [241]
[15] R.N. Dave, R. Krishnapuram, Robust clustering methods: A unified view, IEEE Transactions of Fuzzy Systems 5 (2) (1997) 270-293.
[16] Yihong Dong, Yueting Zhuang, Ken Chen, Xiaoying Tai, A hierarchical clustering algorithm based on fuzzy graph connectedness, Fuzzy Sets and Systems 157 (13) (2006) 1760-1774.
[17] J.J.M. Carrasco, D.C. Fain, K.J. Lang, L. Zhukov, Clustering of bipartite advertiser-keyword graph, in: Proceedings of the Third IEEE International Conference on Data Mining Workshop on Clustering Large Data Sets, 2003.
[18] Z. Broder, S.R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, J. Wiener, Graph structure in the Web, Computer Networks 33 (1-6) (2000) 309-320.
[19] S. E. Virtanen, Clustering the Chilean web, in: Proceedings of the First Latin American Web Congress, LAWEB, IEEE Computer Society, Los Alamitos, CA, USA, 2003.
[20] S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems 30 (1-7) (1998) 107-117.
[21] T.F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoretical Computer Science 38 (1985) 293-306
[22] D.D. Hochbaum, D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, Journal of the ACM 33 (3) (1986) 533-550.
[23] R. Wilson, X. Bai, E.R. Hancock, Graph clustering using symmetric polynomials and local linear embedding, in: British Machine Vision Conference, 2003.
[24] K. Jain, M.N. Murty, P.J.Flynn, Data Clustering: A review, ACM Computing Surveys 31 (3) (1999) 264-323.
[25] F. Aurenhammer, Voronoi diagrams- A survey of a fundamental geometric data structure, ACM Computing Surveys 23 (3) (1991) 345-405.