ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 7, September 2012

GClustering Algorithm

Mr. Promod Kumar Sahu[1], G.Ravi teja[2]

[1]Associate professor, [2]Final M Tech Student

Dept of Computer Science and Engineering

Aditya Institute of Technology and Management – Tekkali, Srikakulam.

***Abstract*—Graph clustering poses significant challenges because of the complex structures which may be present in the underlying data. The massive size of the underlying graph makes explicit structural enumeration very difficult. Consequently, most techniques for clustering multi-dimensional data are difficult to generalize to the case of massive graphs. Recently, methods have been proposed for clustering graph data, though these methods are designed for static data, and are not applicable to the case of graph streams. Furthermore, these techniques are especially not effective for the case of massive graphs, since a huge number of distinct edges may need to be tracked simultaneously. This result in storage and computational challenges during the clustering process.The finding of clusters, well-connected components in a graph, is useful in many applications from natural function prediction to social community detection. An important insight is that many clustering applications need only the subset of best clusters, and not all clusters in the entire graph. In this paper we propose a new technique, GClustering, which probabilistically searches large, edge weighted, directed graphs for their best clusters in linear time. The algorithm is inherently parallelizable, and is able to find variable size, overlapping clusters. To increase scalability, a parameter is introduced that controls memory use. When compared with three other state-of-the art clustering techniques, GClustering algorithm achieves running time speedups of up to 70% on large scale real world datasets. In addition, the clusters returned by GClustering are consistently found to be better both in calculated score and when compared on real world benchmarks.**

***Keywords*— Graph clusters, Hashing, Pruning, Hash Table.**

I. INTRODUCTION

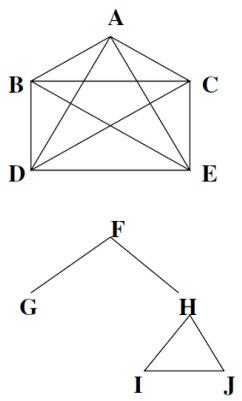
Large amounts of graph data are generated each day from applications such as social networks, communication graphs, biological networks, and more. The structures grow from the hundreds to the millions of nodes and are both useful and important to analyse. As graphs grow in size, it becomes difficult to use or inspect them without some form of summarization. Clustering the nodes of these networks is one technique shown to be of great practical importance. Not only are the small, dense sub graphs easier to visualize and analyse, but well-connected groupings of nodes within graphs have been found to correspond to many real world problems, like biological function prediction and social or web community detection. However, finding the clusters in a graph is difficult as graph sizes grow large. Unfortunately, most current graph clustering algorithms scale poorly in terms of time or memory with increasing graph size.

Several recent graph clustering (and related graph partitioning) techniques have been introduced which improve scalability however, tradeoffs in clustering quality are made, and these techniques are better suited for partitioning graphs into several large pieces rather than finding small densely connected sub graphs. An important study is that all these applications cluster the entire graph, regardless of whether every cluster from the entire graph is needed or useful. As an example, clusters connected weakly, if at all, would be useless for biological function prediction. Applications wish to find significant social groupings would not need clusters returned of people loosely connected. In both these cases, only clusters that are most strongly connected are needed. An algorithm that finds all the clusters of an entire graph, only to rank and keep just a few, is extremely inefficient. Instead, it may be faster and still useful to obtain just the efficient clusters from a large network. In this paper we propose a new algorithm, GClustering, which finds the well connected, clique-like clusters within large graphs. It is naturally parallelizable, and runs in linear time on the graph size.

In addition, the memory consumption can be constrained through input parameters. The algorithm works on both directed and undirected edge weighted graphs, and can also find variable size clusters ranging from an input minimum cluster size to input maximum cluster size. The discovered clusters are allowed to overlap up to a given input percentage. The ability to find slightly overlapping clusters is important in many applications where a single node may be part of multiple clusters, for example in gene networks or social web graphs. This importance has been noted and analysed, however, very few current graph clustering algorithms allow for this feature. The GClustering technique is inspired from Locality Sensitive Hashing (LSH). LSH is a probabilistic method for similarity search, and GClustering is therefore also probabilistic in nature. Well separated clusters with strong, clique-like connections between the nodes are more likely to be found and returned by the algorithm.

The basic idea behind GClustering algorithm comes in two parts. First, many previous clustering techniques and papers have noted that nodes with similar neighbour sets (neighbour hoods) within a graph generally should cluster together. A perfect clique, for example, would contain nodes whose neighbour sets would match exactly between them (all nodes are connected to the same other nodes, if the neighbourhood of a node includes itself). Nodes that do not cluster together would tend to have neighbourhood sets that do not overlap. Fig 1 contains an example which illustrates this graphically. In this figure, five of the nodes form a perfect clique, and

therefore have identical neighbourhoods. The next five nodes do not cluster as well together, and this is reflected in their neighbourhoods. In addition to the neighbourhood, the edge weights among nodes also make a difference in the strength of a cluster.



Node	Neighborhood
A	{A,B,C,D,E}
B	{A,B,C,D,E}
C	{A,B,C,D,E}
D	{A,B,C,D,E}
E	{A,B,C,D,E}
F	{F,G,H}
G	{F,G}
H	{F,H,I,J}
I	{H,I,J}
J	{H,I,J}

Figure 1: Relationship between neighbour hood and clustering. Here, nodes A,B,C,D, and E form a perfect clique, and contain matching neighbour hoods. Nodes F,G,H,I and J do not cluster as well, and this is reflected in their neighbour hoods.

Further, GClustering works in a way so that only those sets of nodes connected by high edge weights are found. It is shown later that the probability of a cluster being found by GClustering is related to the score of the cluster. Second, to reduce the memory consumption needed to cluster a large graph, GClustering takes further advantage of the fact that only the best clusters in the graph need to be found. The algorithm further modifies the LSH process, splitting it into two phases: a pruning phase and a final clustering phase. The pruning phase picks only those nodes most likely to be part of the best clusters, maximizing a score function shown to be related to the final clustering score.

These selected nodes are then clustered in the final clustering phase. Both phases of the algorithm are parallelizable into an arbitrary number of separate threads. This indicates that parallelization frameworks such as Map Reduce can work efficiently with it, which would allow for still further scalability of the GClustering algorithm. The main contributions of this paper include the proposal of a new algorithm, GClustering, which is able to quickly and effectively find strong, overlapping clusters within large, directed, edge weighted graphs. The algorithm is analysed and shown to probabilistically return clusters with higher scores. When implemented and compared with three other state-of-the-art clustering techniques, GClustering is found to achieve running time speedups of up to 70% on large scale real world datasets. In addition, the memory savings achieved from Clustering's pruning algorithm allows it to cluster several massive graphs that other techniques cannot process due to memory constraints. The clusters returned by GClustering are found to score higher both in calculated score and when compared on real world benchmarks. GClustering performs better than the popular clustering tool MCL on biological datasets.

II. Related Work

Several graph clustering algorithms have been proposed, using techniques from areas such as spectral clustering, random walks. Some have been found to return very strong clusters which match well to real world standards and clustering. MCL is a clustering technique based on the simulation of random walk flows which is used in biological networks and returns relevant clusters. By alternating between applying two operators (expansion and inflation) upon the graph matrix of transition probabilities, a clustering related to the random walk distance between nodes can be obtained.

The difficulty with both MCL and the majority of other graph clustering techniques is their inability to scale as graph sizes increase. The complexity of the graph clustering problem on even small graphs, much less on the huge graphs available today, there have been several prior approaches proposed to tackle the scalability bottleneck. One approach is to prune the search space by analysing and returning only the local clustering around a given seed node. This allows the algorithm to focus only on a limited subset of the graph to achieve time and space savings. Examples of such local graph clustering algorithms include the Local Spectral Algorithm and Nibble. Graph partition, which decomposes a graph with a predetermined number of graph cuts, is a separate but related problem to the graph clustering problem we examine here. Graph partitioning can be viewed as clustering of a graph into a predefined number of clusters.

Metis is a multi-level graph partitioning program that achieves scalability by coarsening and performing clustering on a reduced size graph, then uncoarsening to obtain the final partitions. This algorithm, however, restricts its partitions to be nearly equal in size, in addition to the number of partitions being specified in advance. Graclus is another variant on multi-level graph partitioning. By maximizing a weighted k-means objective shown to be equivalent to spectral clustering, Graclus reduces time complexity by avoiding eigenvector computations. In addition, Graclus allows for its partitions to vary in size, though the number must still be specified in advance. An obvious problem with such partitioning approaches is that the number of clusters in a graph is frequently not known in advance, especially for large, complex networks. Furthermore, these algorithms often are most efficient when the number of graph cuts is small, which can lead to increase in cluster sizes for practical analysis. A variation on the MCL algorithm, called RML-MCL that improves MCL's scalability has been recently introduced. RMLMCL replaces the expansion operator with a regularization step. This step allows for a multi-level graph clustering approach to be combined with random walk flow simulations. However, the regularization parameter also leads the algorithm to produce larger clustering, an effect confirmed in our experiments. Local clustering and graph partitioning can both be viewed as methods to reduce the complexity of the graph clustering problem, and allow for a faster, more scalable solution than having to do the global clustering. A different approach, and one that has many practical applications, is to return just the smaller, best scoring clusters

of the entire graph. One recent paper has focused on finding the Efficient-k maximal cliques in uncertain graphs. However, this method of reducing the search space by finding cliques means many useful real world clusters may be missed, as they may connect strongly but be missing several edges between their nodes. Other methods for pruning by finding only the strongest clusters have not yet been explored fully, and this is the problem we focus upon in this paper.

III. Cluster Strength And Scoring

To return only the best clusters in a graph, we need the notion of a score quantifying the strength of a cluster. Lots of dissimilar cluster scoring methods have been introduced over the years, each with their own strengths and weaknesses. The exact choice often depends on the particular application and type of clustering desired. One popular measure of cluster quality is conductance, defined as the ratio of the number of links within a cluster to the number of links leaving that cluster. Conductance has been used in many applications to good results, but it does have several drawbacks. A cluster containing several internally severed components within it may score higher than a similarly sized, better connected cluster of nodes. In addition, conductance as a cluster score does not work well when there are multiple slightly overlapping, but separate, clusters.

In this case, the edges leaving the cluster will fine the score, and often cause the separate clusters to be merged together. A different measure of score is to maximize how clique-like a cluster is. One example is the intra-cluster distance, defined as the ratio between the number of edges in a cluster to the number of possible edges within that cluster. Another related method is the ratio association, a score given to a set of clusters. The ratio association calculates the average link weight within each of its clusters, and then sums these averages. A limitation of both the intra-cluster distance and the ratio association is that there is no preference given to larger cluster sizes (a clique of size 100, for example, is less probable by chance and intuitively should score higher than a clique of size 3). In this paper, since the goal is to find strong, maybe overlapping clique-like clusters, a variation on both the intra-cluster distance and ratio association is used. This score has been found to correlate well with real world clustering. Here, a cluster's score is defined as the average link weight between all nodes in the cluster (including links with weight 0) multiplied by the square root of the cluster size (to allow for a bias towards larger cluster sizes).

IV. Algorithm Overview

To cluster the nodes in a graph, we wish to find those sets of nodes whose neighbourhoods are most similar, and contain high edge weights between them. The GClustering algorithm we propose here makes use of Locality Sensitive Hashing.

Let G={ V,E,w} be a weighted, directed graph where V are the set of vertices, E={(v1,v2) | v1,v2 €V} are set of directed edges and w(v1,v2) (abbreviated as $w_{1,2}$)gives the weight of the edge going from vertex v1to vertex v2. We assume that the edge weights has been normalized such that each link weight is a number between [0, 1] and $\sum_{j\in V,\, j\neq i} w_{i,j}=1$ □ i€ V. In this case, $w_{i,j}=0$ if v_i,v_j .does not belong to E. For simplicity in the later calculations, let n be the number of vertices in the graph. Also let the neighbourhood N_i of vertex v_i be defined as:

Definition1. $N_i= \{ v_i\} U \{ v_j | w_{i,j} > 0\}$ (1)

LSH has been used as a fast and efficient method for linear time similarity search in numerous practical applications..

4.1 Locality Sensitive Hashing (LSH) Applied To Graphs

If we first ignore edge weights, we can use each node's neighbourhood set with a version of LSH that finds similarity based on the Jaccard index to probabilistically find those nodes with similar neighbourhoods. In this technique, each node is represented by a small, length l, probabilistic "signature" created from its neighbourhood.

By generating multiple signatures for each node and hashing them, sets of nodes with matching signatures and similar neighbourhoods may be found. In this section we give a brief overview of this technique. In the two sections following, we modify and extend the LSH algorithm so that it may be used quickly and effectively on graphs.

To create an LSH signature, there are two main steps. First, m random permutations, the nodes in the graph are generated and stored. From this, m "minhash", generated for every node. The value of mhi for a vertex vj is the element in its neighbourhood Nj with the lowest ordering index in _i. So at the end of this step, a node has "m" minhash values, each minhash consisting of one node from its neighbourhood.

There are several limitations with using this algorithm to cluster nodes in a graph.

First, if edge weights are added using methods similar to previously proposed weighting solutions, this would lead to nodes being grouped based on how similar their neighbourhood edge weights are, and not on the strength of the weights.

In addition, another weakness is the low probability of all nodes from a cluster (or even a significant fraction of them) creating the exact same signature. As more nodes are added to a cluster, the probability of them all generating the same signature, unless they are a perfect clique, will decrease. This difficulty has led to different variants of LSH, such as multi-level LSH. In this paper, we introduce a different solution, modification of the hashwords, which fits well with the graph clustering problem we are trying to solve.

Given this weighted neighbourhood, we wish to relate the similarity of two nodes, vi and vj, with their probability of hashing together. To do this, we calculate the probability they will pick the same minhash value.

Raising this probability to the power of the signature length, l, will give their overall chance of hashing together.

For each vertex,v_m€{N_i U N_j} ,there is a 1/ |N_i U N_j| chance that it will be first among N_i U N_j in ordering, for a particular Π. In addition there is a $w_{i,m}$ $w_{j,m}$ chance that it will be included in the neighbourhoods of both v_i and v_j. The

probability that both these will occur and node v_m will therefore be chosen as minhash for both v_i and v_j is :

$$Pr[minhash_{vi,vj}=v_m]= w_{i,m}w_{j,m}/ | N_i \cup N_j| \quad (2)$$

This is the probability that anyone particular vertex V_m will be chosen as the minhash value for both v_i and v_j . there is also the possibility that another vertex $N_i \cup N_j$ was first in ordering but was not contained in the either of the weighted neighbourhood instances leaving V_m to be second in ordering and chosen as the minhash value. And so on for a third, fourth etc.minhash values. However these following probabilities all involve multiplying both the likelihood of matching from Eqn 2 with the likelihood of not matching, raised to the power of the level. These probabilities decrease at an exponential rate. Only the most significant term listed in the Eqn 2.is kept here. The overall probability of v_i and v_j choosing the same minhash(represented by $Pr[minhash_{vi,vj}]$) will be:

$$Pr[minhash_{vi,vj}]=\sum_{K\in(Ni \cup Nj)} w_{i,k}w_{j,k}/ | N_i \cup N_j | \quad (3)$$

Note that we taking the union in the summation and therefore some weights may be zero. From eqn 3.we can already see intuitively that the probability of the two nodes hashing together is maximized not only when their neighbourhoods are similar(both $w_{i,k}$ and $w_{j,k}$ are nonzero for the same nodes v_k), but also when the weights are maximized. Extending eqn 2 to the probability of cluster of nodes $C=\{v_1,v_2,\ldots v_c\}$ gives following probability for all nodes in C to pick a particular node v_k as minhash:

$$Pr[minhash_c= V_k]= \Pi_{i\in C} w_{i,k}/|U_{i\in C}N_i| \quad (4)$$

Eqn 4 may be rearranged to solve for $\Pi_{i\in C} w_{i,k}$, which gives the following equation:

$$\Pi_{i\in C} w_{i,k}= Pr[minhash_c= V_k]. |U_{i\in C}N_i| \quad (5)$$

The left side of this equation is taking the product of a series of numbers. This product can be related to the sum of the same series according to the relation observed in [16].

OBSERVATION: if we are given a positive real number k, and a fixed positive integer n,then among all of the real factor sets of k that consists of n numbers,the real factor set that yeids the minimal sum consists of n copies of $K^{1/n}$.
This means equation 5 can be converted to the following relation

$$\sum_{i\in C} w_{i,k} \geq |C|(Pr[minhash_c= V_k]. |U_{i\in C}N_i|)^{1/|c|} \quad (6)$$

Examining equation 1,our equation for scoring a cluster,we can rearrange the summations in the numerator to form $\sum_{i\in C} \sum_{i\in C}$ wi,j. From this we can substitute equation 6 with the $\sum_{i\in C}$ wi,j term.

4.3 Pruning of Search Space

A weakness of the above hashing technique is the memory requirement. Every node in the graph will be stored, along with its hash word, w number of times. With each hash word of length l, the memory consumed will be O(n_w_ l). When working with very large graphs, it is desirable to decrease memory requirements to below O(n). Since GClustering need to search only for the efficient clusters in a graph, this allows for greater pruning of the search space. We can therefore further modify the LSH process, splitting it into two parts. First, a smaller low memory version of the modified LSH function is run on every node to pick out the efficient p nodes in the graph (nodes most likely to be part of the best clusters). Next, the full version of the algorithm described in the preceding sections is run on only these efficient p nodes. In this way the memory consumption is now restrained to O(p_w_ l). This low memory version is based on the idea that a "promising" node is one most likely to increase the score of a high scoring cluster. To discover this likelihood, we first notice that, given a vertex va, the nodes of any cluster, C = fv1; v2; vcg containing va can be divided into three subsets. Node va itself, the cluster nodes neighbouring va in the set S = C \ Na, and the remaining set of nodes, R such that fvag [S [R = C]]. An illustration of this division is shown in Fig 2. For a high scoring cluster, jSj and the

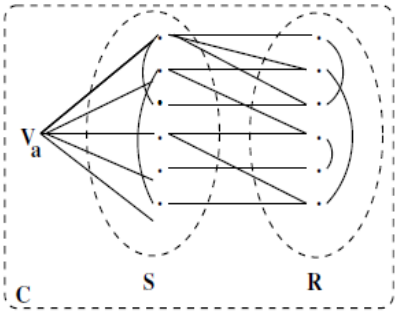


Figure 2: Illustration of subsets v_a, S, and R in cluster C

The larger each of these sets are, the better the overall cluster score. Maximizing all parts of this score would lead to obtaining clusters with maximum score, and therefore requires clustering the entire graph. To minimize time complexity, we choose to maximize just the first three terms as an approximation, obtaining a subscore related to the maximal scoring of the overall cluster. Experimental results confirm that this subscore is effective at choosing nodes likely to participate in strong clusterings, as discussed later in Section

Score (VS + SV + SS + RR)

In addition, the list of possible nodes in S is limited to those connected to va with highest edge weights. Given that the maximum size of a cluster was a parameter Smax, we need only to consider some multiple of the Smax Efficient nodes in Na to get a notion of the likelihood of va being in a Efficient cluster. (If the nodes in S are not contained within these Efficient neighbors, then either a significant number of high weighted links leave the cluster and it is not well separated, or the links are all of low weight and va is not likely to participate in a high scoring cluster). This will give us a set, Sa, of nodes.

V. CONCLUSIONS

Graph clustering is an important tool for the mining and visualization of graphs, especially in those massive graphs that are most difficult to handle. Though finding all clusters in a massive graph may take an inordinate amount of time and space, it is possible to prune the search space by examining just for those clusters with highest scores. Since many real world applications on large graphs need only the subset of most strongly connected clusters, this is a solution that can produce both interesting and relevant results. In this paper we have introduced a new algorithm named as GClustering, which allows for the probabilistic search of a graph for its efficient scoring clusters in linear time. We also showed that the probability of GClustering finding a cluster is related to a lower bound on the cluster score.

GClustering works with directed or undirected, weighted graphs, and finds clusters that overlap to a given percentage. It has a limited memory consumption, and testing GClustering against other state-of-the-art graph clustering techniques shows GClustering consistently performing faster, scaling well with graph size, and performing up to 70% faster on real world datasets. When scaled to massive real world graphs of up to 5 million nodes and 70 million edges, all three other previous graph clustering techniques were unable to run to completion, due to their time and memory constraints. However, GClustering is able to return high scoring clustering on these massive datasets with just 1GB-2GB of resident set memory and completes in a running time of a few minutes.

Finally, the efficient clusters returned by the GClustering algorithm consistently have higher scores than other current clustering techniques, when compared to both calculated scores and real world benchmarks. GClustering algorithm is shown to return up to 11% more clusters with higher real world biological relevance than MCL, a popular graph clustering algorithm in bioinformatics. These results lead us to conclude that GClustering technique for pruning the search space on large graphs to find efficient scoring clusters is an effective solution to this interesting problem, and hence the GClustering algorithm is both useful and relevant as a scalable clustering solution on massive real world graphs.

REFERENCES

[1] A.Abourjeili and G. Karypis. Multilevel algorithms for partitioning powerlaw graphs. In IPDPS, pages 16–575, 2006.
[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In FOCS, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society.
[3] M. Ashburner and et. al. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nat Genet, 25(1):25–29, May 2000.
[4] B. Andreopoulos, A. An, X. Wang, and M. Schroeder. A roadmap of clustering algorithms: finding a match for a biomedical pplication. Brief Bioinform, 10(4):297–314,May 2009.
[5] G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics, 4(2), 2003.
[6] Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. Comput. Netw. ISDN Syst.,29(8-13):1157–1166, 1997.
[7] S. Brohee and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinformatics, 7:488, November 2006.
[8] Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In VLDB, pages 518–529,1999.
[9] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web (extended abstract). In WebDB, 2000.
[10] P. Indyk. A small approximately min-wise independent family of hash functions. In SODA '99, pages 454–456, 1999.
[11] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. J. ACM, 51(3):497–515, May 2004.
[12] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In In SDM, 2004.
[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. pages 137–150.
[14] S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. TPAMI, 29(11):1944–1957, 2007.
[15] S. Fortunato. Community detection in graphs. Physics Reports, 486:75–174, Feb. 2010

G.Ravi teja, doing M.Tech(CSE) in Aditya Institute of Technology and Management (JNTUK), Tekkali, Andhra Pradesh, India. His research interest includes Data mining and ware housing.

Mr **Promod Kumar Sahu**, M.Tech(CSE) working as Associate Professor in Computer Science and Engineering department, Aditya Institute of Technology and Management, Tekkali, Andhra Pradesh, India. He has 11 years of experience in teaching Computer Science and Engineering related subjects. He has published several research papers in national and international journals/conferences. He has guided more than 70 students of Bachelor Degree and 20 students of Master Degree in Computer Science and Engineering in their major projects. He is a member of ISTE and CSI.