Fast Incremental Minimum-Cut Based Algorithm for Graph Clustering

Barna Saha
Indian Institute of Technology, Kanpur
barna@cse.iitk.ac.in

Pabitra Mitra
Indian Institute of Technology, Kharagpur
pabitra@iitkgp.ac.in

Abstract

In this paper we introduce an incremental clustering algorithm for undirected graphs. The algorithm can maintain clusters efficiently in presence of insertion and deletion (updation) of edges and vertices. The algorithm produces clusters that satisfies the quality requirement, given by the bi-criteria of [6]. To the best of our knowledge, this is the first clustering algorithm, for dynamic graphs, providing strong theoretical quality guarantee on clusters. The incremental clustering algorithm, is motivated by the minimum-cut tree based partitioning algorithm of [6] and [7]. It takes $O(k^3)$ time for each updation, where k is the maximum size of any cluster. This is the worst case time complexity, and in general time taken is much less. In presence of only insertion of edges and vertices, or for graphs stored in secondary storage, the algorithm takes, $O(mk^3)$ time for clustering, in contrast to $O(n^3)$ time requirement of [6]. Here m and n are respectively the number of edges and vertices of the graph. Many real world graphs, have been found to have low average degree, for which $n = \Theta(m)$. For these classes of graphs, even when the graph is stored, the proposed algorithm performs far better than [6], in terms of computation time. This has been validated by our experimental results on several benchmark test graphs upto 30 thousand nodes and 1 million edges, obtained from varied application areas. Finally, a comparison has been drawn between the proposed algorithm and a recent multi-level kernel based algorithm for fast graph clustering [5], and the effectiveness of our algorithm is established.

1 Introduction

Clustering, or partitioning of data in natural groups, is a fundamental problem in computer science, mathematics and many applied areas. Good clustering, groups similar items in a cluster, and data in different clusters exhibit dissimilar properties. The objective of many clustering algorithms is typically to optimize certain statistical measures. Algorithms in this category are—k-means, k-median, minimum-sum, minimum-diameter etc.. A related and very important class of clustering algorithms deal with graphs. The goal is to create clusters that are heavily connected, in addition to, assuring that inter-cluster connectivity is low. In light of this, Kannan et al. [10] suggested a bi-criteria measure for good clustering, where clusters satisfy certain expansion or conductance (See section 2) measure and weight of the edges across the cluster is upper bounded. A slight variant of the above measures is given in [6], where instead of inter-cluster edge weight, expansion across the clusters is maintained below a threshold. In reality, both these measures perform well.

1.1 Previous Work on Graph Clustering

There exists a rich literature on graph clustering algorithms. Among them one of the most popular methods is spectral clustering. It deals with the matrix structure of the graph and partitions the rows into few eigen-vectors [10, 16], depending on their components. Graph clustering methods have also been proposed using random walks, based on Markov chain [3]. The rapid mixing of Markov chain is a necessary condition for producing good clusters in this method. This requires that the second eigen-value of the adjacency matrix be well-separated from the largest eigen value one. In general there is no guaranteed bound on the mixing time and hence the procedure is not suitable for time-critical applications. Spectral-method though suffices the condition of bounded polynomial time complexity, but it requires processing of the entire graph together. So when graphs are originated from real world applications, like WWW, Internet, Mobile Networks, etc., the massive size of these graphs makes the space-complexity of the spectral algorithm prohibitive. A slight change in the edge-relations necessitates re-processing of the entire graph structure. Recently a fast algorithm for graph clustering has been suggested in [5] on the line of multi-level partitioning scheme of METIS [11]. The procedure involves repeated coarsening of the original graph to smaller size graph, partitioning it and then remapping the partitions back. However it does not say, how to modify the clusters under mobile scenario, when edges and vertices may get added or deleted arbitrarily. The algorithm cannot provide any theoretical guarantee on the clustering quality. A new direction to graph-clustering has been introduced by modeling the clustering problem, using minimum-cut, maximum-flow problem of the underlying graph. Work on this approach [15, 7] has been used to identify web-communities, to segment images etc.. Flake etal. has used this method to produce clusters [6], with theoretical quality measure, that works remarkably in practice [6, 12]. In addition to its wide application on clustering graphs, it has also been used as learning algorithm [14]. However the algorithm requires processing of the entire graph, everytime the graph structure undergoes some change. Therefore the algorithm of [6] is not suitable for clustering dynamic graphs.

1.2 Dynamic Graphs and Incremental Graph Clustering

The World Wide Web (WWW) can be viewed as a massive graph, where nodes are the pages and an edge between two pages, represents a link between them. Finding web-communities using link structure of the WWW is an important problem [7]. However the highly dynamic nature of the web renders all the above discussed methods for clustering useless. We need clustering that can efficiently process, insertion and deletion of nodes and links, without requiring to recluster the entire graph at every alteration.

Consider Citeseer data base[1] [6], where each research publication can be modeled by a node and an edge from publication 1 to publication 2 represents citation of publication 2 by publication 1. Clustering over this citation graph enables detecting research on related topics. However this graph is continuously changing, with new publications being added at every instant and old out-of-date publications being removed. Therefore we need clustering algorithms that can dynamically adjust with this changing environment.

Clustering nodes in ad hoc sensor networks [2] has also been very useful for improving network lifetime, scalability, load-balancing. Routing in mobile networks can be made more efficient using clustering information. The basic objective is to select a node from each cluster to act as cluster-head. The cluster head is responsible for coordination among nodes within the cluster and for communication across the clusters. However all these networks are highly mobile, and hence the traditional static graph clustering algorithms cannot be applied here. Some clustering schemes have been specifically devised to suit these applications. Recent research in this area include HEED [1], LEACH [1], ACE [4], DCE [2]. But in all these algorithms, cluster-heads remain static. Any alteration in cluster-heads triggers clustering of the whole graph afresh. The quality of clusters is experimentally verified, without any theoretical guarantee. The algorithms do not meet the quality requirement of either [10] or [6]. In addition, HEED and LEACH require $O(n)$ (n is the number of nodes) processing time for each node.

Clustering is also useful for resource allocation in mobile network. It provides spatial reuse of bandwidth due to node clustering. Bandwidth can be effectively handled and shared within each cluster. However we need clustering algorithm that is robust over topological changes due to node movement, node failure, node insertion and deletion. [13, 9] describe clustering algorithms for mobile radio network. However the algorithms do not give theoretical guarantee on clustering quality, neither they experimentally verify quality requirements given by the bi-criteria of [10] or [6].

The importance of developing incremental clustering algorithms, for dynamic networks, that can be deployed in these diverse application fields, that is theoretically strong and experimentally verified, is therefore unquestionable. The algorithm should process each update, in the form of addition and removal of nodes or links, in time and space much less than n. At any instant of time, the existing clusters can be obtained easily. The clusters must satisfy theoretical quality guarantee and should be verified to work well in practice. In this paper we develop a fast incremental graph clustering algorithm, satisfying the above criterias.

1.3 Contribution

Our contributions are as follows:

- We develop an incremental graph clustering algorithm that can handle insertion and deletion of edges, while clustering is on progress. Each update operation requires time $O(k^3)$ (worst case). This can be reduced to $O(k^2)$, using a heuristic. Here k is the maximum size of any cluster, which is much less than the total size of the network, generally logarithmic of the total size. Vertex addition and deletion can also be handled efficiently. At any instant, the algorithm maintains the clusters of the existing graph. If links and nodes can only be added, the clustering can be performed in time $O(mk^3)$. If k is logarithmic of n, then since most of the real-world networks have constant average degree [6], the time taken to perform clustering is $O(n(\log n)^3)$.
- A detailed theoretical analysis of the clustering quality is provided.
- The effectiveness of the algorithm is established through experimentation on several benchmark graphs, upto 30 thousand nodes and 1 million edges. Comparison results with the algorithm of [6] and a cutting-edge

[1]www.citeseer.com

multilevel algorithm of [5] clearly demonstrates the superiority of our incremental algorithm.

1.4 Organization

The rest of the paper is organized as follows. Section 2 delineates the clustering algorithm based on minimum-cut tree developed in [6]. This algorithm serves as the basis for our incremental algorithm. Section 3 describes our proposed incremental graph clustering algorithm and gives detailed proof of its quality guarantee. Section 4 shows the efficacy of our method with detailed experimentation . Finally we conclude in Section 5.

2 Clustering Using Mincut Tree

2.1 Preliminaries

Let $G = (V, E)$, denote a weighted undirected graph with $n = |V|$ nodes or vertices and $m = |E|$ links or edges. Each edge $e = (u, v)$, $u, v \in V$ has an associated weight $w(u, v) > 0$. The adjacency matrix A of G is an $n \times n$ matrix in which $A(i, j) = w(i, j)$ if $(i, j) \in E$, else $A(i, j) = 0$. The corresponding adjacency list structure maintains for every $i \in V$ only those j's, for which $w(i, j) > 0$. The $w(i, j)$ values are also retained.

Let s and t be two nodes in $G(V, E)$, designated as source and destination respectively. The minimum cut of G with respect to s and t is a partition of V, namely, S and V/S, such that $s \in S$, $t \in V/S$ and total weight of the edges linking vertices in two partitions is minimum. The sum of the edge-weights across S and V/S, is denoted by the cut-value, $c(S, V/S)$. S is called the community of s. The minimum cut tree, T_G of G, defined in [8], is a tree on V, such that inspecting the path between s and t in T_G, the minimum-cut of G with respect to s and t can be obtained. Removal of the minimum weight edge in the path yields the two partitions and the weight of the corresponding edge, gives the cut-value.

2.2 Clustering Algorithm

[6] defines clustering based on the minimum-cut tree. An artificial sink, t, is added in the graph and is connected to all the vertices. Each edge (t, v), $v \in V$ has the associated weight $\alpha > 0$. The value of α is critical in determining the quality of the clusters. The minimum-cut tree is then computed on this new graph. The disjoint components obtained after removal of t from the minimum-cut tree are the required clusters. The algorithm is named as "Cut-Clustering algorithm". Figure 1 shows the communities of the minimum-cut tree after addition of the artificial sink t. Fig 2 gives the basic "Cut-Clustering algorithm".

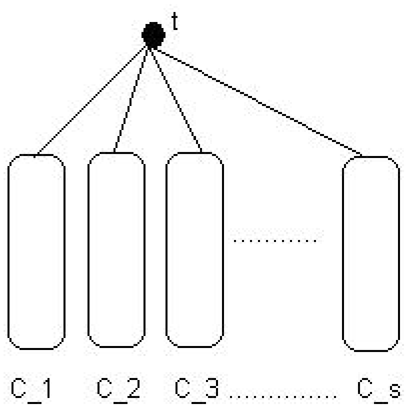

Figure 1. Communities of the Minimum-Cut Tree

```
Cut-Clustering($G(V, E)$,$\alpha$)
begin
    Let $V$ := $V \cup t$
    For all vertices $v$ in $G$
        Connect $t$ to $v$ with edge of weight $\alpha$
    Let $G'(V', E')$ be the new graph after connecting $t$ to $V$
    Calculate the minimum-cut tree $T'$ of $G'$
    Remove $t$ from $T$
    Return all connected components as the clusters of $G$
end
```

Figure 2. Cut Clustering Algorithm of [6]

2.3 Clustering Quality

The quality of the clusters produced using "Cut-Clustering" algorithm, is measured using expansion like criteria. Let $(S, \bar{S})$ be a cut in G. The expansion of this cut is defined as

$$\begin{aligned} \Psi(S) &= \frac{\sum_{i \in S, j \in \bar{S}} w(i, j)}{min\{|S|, |\bar{S}|\}} \\ &= \frac{c(S, \bar{S})}{min\{|S|, |\bar{S}|\}} \end{aligned}$$

The expansion of a (sub)graph is the minimum expansion over all the cuts of the (sub)graph. Higher the expansion of a cluster, better is its quality.

[6] assures that if S is a cluster produced by the "Cut-Clustering" algorithm, then the following conditions are

satisfied (See Fig 3):

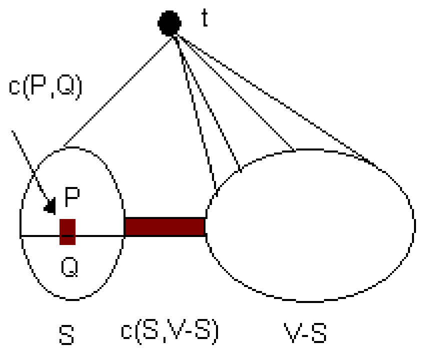

Figure 3. Inter and Intra Cluster Cuts

1. $\frac{c(P,Q)}{min\{|S|,|\bar{S}|\}} \geq \alpha$, for any $P, Q \subset S$, such that $P \cup Q = S$ and $P \cap Q = \phi$.
2. $\frac{c(S,V-S)}{|V-S|} \leq \alpha$.

Therefore α serves as a lower bound for intra-cluster expansion and an upper bound for inter-cluster expansion. This clustering quality measure is not exactly same as the bicriteria proposed by Kannan et.al [10], but closely similar to it and has its own advantage.

3 Incremental Clustering with Min-Cut Tree

In this section we present our main contribution—an incremental algorithm for graph-clustering. The clustering technique is motivated by the “Cut-Clustering” algorithm described in the previous section.

3.1 Intuition

Intuitively, it appeals that a small change in the edge relationship of a graph should not disturb the existing clusters immensely. The clusters of the new graph should be generated efficiently from the knowledge of the previous clusters without reprocessing the graph in its entirety. Addition of intra-cluster edges or deletion of inter-cluster edges improves the connectivity within the existing clusters. Hence under these conditions, it is quite natural for the new updated graph to have the same clusters, unless the previously existing clusters were not satisfactory. On the contrary inter-cluster edge addition and intra-cluster edge deletion degrades the quality of the existing clusters and the new graph is likely to have different clusters. But it is very unlike that all the clusters would be affected. Only those clusters which have suffered changes in the form of edge addition or deletion are likely to produce different clusters in the altered scenario. Processing only these few clusters and keeping the others unchanged should alone be sufficient to produce quality clusters in the updated graph.

3.2 Algorithm

Our proposed algorithm maintains for every vertex, two variables, *In Cluster Weight (ICW)* and *Out Cluster Weight (OCW)*. Let $C_1, C_2, ..., C_s$, $s > 0$ is an integer, be the clusters of $G(V, E)$. Then *ICW* and *OCW* are defined as below.

Definition 1 ***In Cluster Weight (ICW).*** In Cluster Weight *or* ICW *of a vertex $v \in V$ is defined as the total weight of the edges linking the vertex v to all the vertices which belong to the same cluster as v. That is, if $v \in C_i$, $0 \leq i \leq s$, then $ICW(v) = \sum_{u \in C_i} w(v,u)$.*

Definition 2 ***Out Cluster Weight (OCW).*** Out Cluster Weight *or* OCW *of a vertex $v \in V$ is defined as the total weight of the edges linking the vertex v to all the vertices which do not belong to the same cluster as v. That is, if $v \in C_i$, $0 \leq i \leq s$, then $OCW(v) = \sum_{u \in C_j, j \neq i} w(v,u)$.*

The main feature of our algorithm is that it builds part of the minimum-cut tree as and when necessary. Minimum-cut tree is computed only over a very small graph, created efficiently from the original graph. This has a flavor of coarsening step of [5]. However no remapping or refinement step is necessary. Clusters of the original graph can be directly obtained from the coarsened graph. The algorithm is thus very fast. At any instant, the set of clusters of the graph, seen so far, can be obtained without any further processing. The cluster quality is identical to the offline “Cut-Clustering” algorithm (See Figure. 2).

The algorithm supports and maintains clusters over the following four update operations.

1. Edge Insertion.
2. Edge Deletion.
3. Vertex Insertion.
4. Vertex Deletion.

Edge insertion and deletion involve edges, where end vertices have already been seen.

Let $\mathcal{C} = \{C_1, C_2, ..., C_s\}$, are the clusters of the graph $G(V, E)$, that has been seen so far. Let A be the adjacency matrix of G. Below we give description of our algorithm over the four update operations.

1. Edge Insertion.

Intra-Cluster Edge Insertion.

When an edge gets added, whose both end vertices belong to the same cluster, we simply update the adjacency matrix and *ICW*. The clusters remain unchanged. Figure 4 gives the algorithm for *intra-cluster edge addition*. The time required is $O(1)$.

```
Intra-Cluster Edge Addition($(i,j)$,$w_{i,j}$)
begin
    Let $i,j \in C_u$
    Update $A(i,j) += w(i,j)$
    Update $ICW(i) += w(i,j)$, $ICW(j) += w(i,j)$
    Return $\mathcal{C}$
end
```

Figure 4. Intra-Cluster Edge Addition

Inter-Cluster Edge Addition. Addition of an edge, whose end points belong to different clusters increases connectivity across the clusters. Therefore as a result, the clustering quality suffers. If the quality measure, given in Subsection 2.3, is not maintained, re-clustering becomes necessary. To understand the algorithm in the case of *inter cluster edge addition*, we need to first look into two processes, *merging of clusters* and *contraction of clusters*.

```
MERGE($C_u$,$C_v$)
begin
    $D = C_u \cup C_v$
    For all $u \in C_u$
        Update $ICW(u) += \sum_{v \in C_v} w(u,v)$
        Update $OCW(u) -= \sum_{v \in C_v} w(u,v)$
    For all $v \in C_v$
        Update $ICW(v) += \sum_{u \in C_u} w(v,u)$
        Update $OCW(v) -= \sum_{u \in C_u} w(v,u)$
    Return D
end
```

Figure 5. Merging of Two Clusters

Merging of Clusters. Merging of two clusters C_u and C_v is described in Figure 5. By merging, a single cluster is formed containing the vertices of C_u and C_v. ICW and OCW can easily be updated using the adjacency matrix of the graph G. The time complexity for merging is $\Theta(|(|C_u) + |C_v|) = \Theta(|C_u + C_v|)$.

Contraction of Clusters. Contraction of $A \subset V$ in G is performed by replacing A with a single node x. Self loops, resulting from the edges connecting vertices in A, are removed. Parallel edges are replaced by a single edge, having weight equal to the sum of the weights of the parallel edges. While contracting clusters, A represent a single or multiple clusters. The process to contract clusters is described in Figure 6. The contracted graph can be obtained from the adjacency matrix of the original graph along with *ICW* and *OCW* in time $\Theta(|A|^2)$.

```
CONTRACT($G(V,E)$,$S$)
Comment. $S$ is a set of clusters
begin
    Let $A'$ denote the adjacency matrix of the contracted graph
    $V' = \{V - S, x\}$, $n' = |V'|$
    Copy the entries of $A$, involving both the vertices
    from $V - S$, to $A'$
    $A'(i,n) = ICW(i) + OCW(i) - \sum_{j=1}^{n-1} A'(i,j)$
    Comment. $E'$ can be obtained from $A'$
    Return $G'(V',E')$
end
```

Figure 6. Creating Contracted Graph

Now we are ready to describe our algorithm for *inter-cluster edge addition* (Figure 7). If the addition of edge does not deteriorate the clustering quality (Condition 1), then the same clusters are maintained. Else if, Condition 2 is satisfied, then the two clusters, C_u and C_v, containing the end-vertices of the inserted edge, are merged. Otherwise (Condition 3), we create a coarsened graph, by contracting all the clusters except C_u and C_v to x. The resulting graph has only $|C_u + C_v| + 1$ vertex entries and significantly smaller than the original graph. Similar to "Cut-Clustering" algorithm, we add an artificial sink t and add edges connecting t to all vertices in the coarsened graph. However, the weight of the edge linking t to x is $|V - C_u - C_v|\alpha$. All other edges with t as one end-point, have weight of α. The minimum-cut tree is computed over this graph. The connected components are computed from this tree, after removing t. Those components containing vertices of C_u and C_v along with the clusters $\mathcal{C} - \{C_u, C_v\}$ are returned as the clusters of the original graph. The entire process takes time $\Theta(|C_u + C_v|^3)$. Using a heuristic (Subsection 3.5) the processing time can be reduced to $\Theta(|C_u + C_v|^2)$.

2. Edge Deletion

Intra-Cluster Edge Deletion. When an edge gets deleted inside a cluster, the connectivity within cluster deteriorates, arising the need of reclustering. The case is handled in a similar fashion as in *inter-cluster edge addition* for Condition 3. Here the end vertices of the edge deleted, belongs to the same cluster C_u. So except C_u, all the other clusters are contracted to form the coarsened graph. The

```
Inter-Cluster Edge Addition($(i,j)$,$w(i,j)$,$\alpha$)
begin
    Let $i \in C_u$ and $j \in C_v$
    If $\frac{\sum_{u \in C_u} OCW(u)+w(i,j)}{|V-C_u|} \leq \alpha$ and
$\frac{\sum_{v \in C_v} OCW(v)+w(i,j)}{|V-C_v|} \leq \alpha$ (CONDITION 1)
    Then
        Update $A(i,j)+=w(i,j)$
        Update $OCW(i)+=w(i,j)$, $OCW(j)+=w(i,j)$
        Return $C$
    Else
        If $\frac{2c(C_u,C_v)}{V} \geq \alpha$ (CONDITION 2)
        Then
            D=MERGE($C_u, C_v$)
        Return $C+D-\{C_u,C_v\}$
        Else {(CONDITION 3)}
            $G'(V',E')$ =CONTRACT($G(V,E)$,$V-C_u-C_v$)
            Connect $t$ to $v$, $\forall v \in C_u, C_v$ with edge of weight
            $\alpha$
            Connect $t$ to $V'-\{C_u,C_v\}$ with edge of weight
            $\alpha|V-C_u-C_v|$
            Let $G''(V'',E'')$ is the resulting graph
            Calculate MINIMUM-CUT Tree $T''$ of $G''(V'',E'')$
            Remove $t$
            Let$\{D_1,D_2,..,D_k\}$, $k>0$, are the connected
            components of $T''$ after removing $t$, containing
            vertices of $C_u$ and $C_v$.
            $C=\{D_1,D_2,..,D_k,C_1,C_2,..,C_s\}-\{C_u,C_v\}$
            Return $\mathcal{C}$
end
```

Figure 7. Inter-Cluster Edge Addition

time complexity to process the update is $\Theta(|C_u|^3)$, which can be reduced to $\Theta(|C_u|^2)$ using the heuristic (Subsection 3.5). The detailed algorithm is given in Figure 8.

Inter Cluster Edge deletion. When an edge with end points in two different clusters, gets deleted, the connectivity across the clusters becomes less. As a result, the clusters become more well-connected. Hence in this case, we return the existing clusters after updating the adjacency matrix of the graph and *OCW* (See Figure 9). The time taken by the process is $O(1)$.

3. Vertex Addition. Addition of new vertices is handled efficiently as below:

- If an isolated vertex gets added, it is kept as a separate cluster.

- If an edge (i,j) arrives, where the vertex i has appeared earlier but the vertex j is new. Then j is treated as a separate cluster and the algorithm processes the edge as an instance of an *inter cluster edge addition*.

- If both i and j have not appeared before, then depending upon the weight, $w(i,j)$, the edge is processed. If $w(i,j) \geq \alpha$, a new cluster is formed containing only i and j. Else two new clusters are created containing singleton vertices i and j respectively.

```
Intra-Cluster Edge Deletion($(i,j)$,$w(i,j)$)
begin
    $A(i,j)-=w(i,j)$
    Let $i,j \in C_u$
    $G'(V',E')$ =CONTRACT($G(V,E)$,$V-C_u$)
    Connect $t$ to $v$, $\forall v \in C_u$ with edge of weight
    $\alpha$
    Connect $t$ to $V'-C_u$ with edge of weight
    $\alpha|V-C_u|$
    Let $G''(V'',E'')$ be the resulting graph
    Calculate MINIMUM-CUT Tree $T''$ of
    $G''(V'',E'')$
    Remove $t$
    Let$\{D_1,D_2,..,D_k\}$, $k>0$, are the connected
    components of $T''$ after removing $t$, containing
    vertices of $C_u$
    $C=\{D_1,D_2,..,D_k,C_1,C_2,..,C_s\}-\{C_u\}$
    Return $\mathcal{C}$
End
```

Figure 8. Intra Cluster Edge Deletion

4. Vertex Deletion.

- If an isolated vertex gets deleted, then that vertex entry is removed from the adjacency matrix and nothing else is done.

```
Inter-Cluster Edge Deletion($(i,j)$,$w_{i,j}$)
begin
    Let $i \in C_u$ and $j \in C_v$, $u \neq v$
    Update $A(i,j)-=w(i,j)$
    Update $OCW(i)-=w(i,j)$,
           $OCW(j)-=w(i,j)$
    Return $C$
end
```

Figure 9. Inter-Cluster Edge Deletion

- Else, let the cluster which contains the vertex v to be deleted, be C_v. All edges, (u,v), where $u \notin C_v$ are deleted and the adjacency matrix along with *ICW* and *OCW* are updated similar to *Inter Cluster Edge Deletion*. Rest of the edges emanating from v are removed and the resulting graph is processed, following the procedure of *Intra Cluster Edge Deletion*. Finally v is removed from C_v and its entry from the adjacency matrix is discarded.

3.3 Time Complexity

Let $k = max_{u=1}^{s}\{|C_u|\}$. Then the time complexity for insertion of inter-cluster edge and deletion of intra-cluster edge is $O(k^3)$. Processing intra-cluster edge addition and inter-cluster edge deletion, require $O(1)$ time. Vertex addition and deletion can both be handled in $O(k^3)$ time. Therefore update-processing time is $O(k^3)$. In contrast, the offline ”Cut-Clustering algorithm” has no mechanism to maintain clusters over dynamic graphs. It requires time $O(n^3)$ for each update processing. If a graph is stored in secondary memory, we can start from empty clusters and process each edge of the graph sequentially. Time required to process the entire graph is $O(mk^3)$ (actually much less than this). Generally the clusters are small, compared to the total number of vertices. Therefore if $k = O(\log n)$, we have an $O(m(logn)^3)$ algorithm. Most of the massive graphs that occur in real life have very low average degree. For example, citeseer citation graph has an average degree of 3.5 [6]. So for these classes of graphs, $n = \Theta(m)$ and we have an $O(n\, polylog(n))$ algorithm to obtain clustering, which is much better than the $O(n^3)$ time requirement of the offline “Cut-Clustering” algorithm.

3.4 Proof of Clustering Quality

In this section, we show that the clusters obtained by our incremental algorithm, has the same quality guarantee of the offline clustering algorithm of [6]. Precisely we show,

1. $\frac{c(P,Q)}{min\{|S|,|\bar{S}|\}} \geq \alpha$, for any $P, Q \subset S$, such that $P \cup Q = S$ and $P \cap Q = \phi$.
2. $\frac{c(S,V-S)}{|V-S|} \leq \alpha$.

We show that the two above criteria are satisfied over all the updates. Since we start clustering with an empty graph and at every update maintain these criteria, by induction the proof follows.

1. Edge Addition.

Intra-Cluster Edge Addition.

Let the edge added be (i,j), $i,j \in C_u$. $\frac{c(C_u,V-C_u)}{V-C_u} \leq \alpha$ because, $c(C_u, V-C_u)$, remains unaltered . Offcourse, for $P,Q \subset C_v$, $v \neq u$, $\frac{C(P,Q)}{min\{|P|,|Q|\}}$, cannot change. For $P,Q \subset C_u$, if $i,j \in P$ or $i,j \in Q$, $c(P,Q)$ remains unchanged. Consider among those partitions P,Q of C_u, in which $i \in P$ and $j \in Q$ and for which $\frac{c(P,Q)}{min\{|P|,|Q|\}}$ is minimum. This value is greater than α. Addition of the edge, (i,j), increases $c(P,Q)$ to $c(P,Q) + w(i,j)$. Hence the ratio $\frac{c(P,Q)}{min\{|P|,|Q|\}}$ increases, improving the clustering quality.

Inter-Cluster Edge Addition.

CONDITION 1. Note that, $\sum_{u \in C_u} OCW(u) = C(C_u, V-C_u)$. Therefore if CONDITION 1 is satisfied, then for all u, $\frac{C(C_u,V-C_u)}{V-C_u} \leq \alpha$. Under this condition, the previous clusters are returned by the algorithm. Therefore by quality guarantee of the existing clusters, $\frac{C(P,Q)}{min|P|,|Q|} \geq \alpha$.

CONDITION 2. Lemma 1 is obtained from the property of the minimum-cut tree. Lemma 2 establishes the quality guarantee of our algorithm under CONDITION 2, using Lemma 1.

Lemma 1 *Let T be the minimum-cut tree of G, after addition of the artificial sink t. If P,Q, $P \neq \phi$, be any cut of a connected component, S, of T after removing t, then $c(x,Q) \leq c(P,Q)$, where x is obtained by contracting $t \cup X$ in T.*

Proof. See Lemma 3.2 of [6]. □

Lemma 2 *If $\frac{2c(C_u,C_v)}{|V|} \geq \alpha$ then merging of C_u and C_v, maintains the clustering quality.*

Proof. Let $D = C_u \cup C_v$. For all $i \neq u, v$, $c(C_i, V-C_i)$ remains unchanged. We see

$$\begin{aligned} & c(D, V-D) \\ = \; & c(C_u, V-C_u) + c(C_v, V-C_v) - 2c(C_u,C_v) \\ \leq \; & \alpha(|V-C_u|) + \alpha(|V-C_v|) - 2c(C_u,C_v) \\ = \; & \alpha(|V-C_u+C_v|) + \alpha|V| - 2c(C_u,C_v) \end{aligned}$$

If $\alpha|V| \leq 2c(C_u,C_v)$, or $\alpha \leq \frac{2c(C_u,C_v)}{|V|}$, then $\frac{c(D,V-D)}{|V-D|} \leq \alpha$.

Now,

$$\frac{c(C_u,C_v)}{\min|C_u|,|C_v|} \geq \frac{2c(C_u,C_v)}{|V|} \geq \alpha$$

Let $P,Q \subset D$, $P \cup Q = D$ and $P \cap Q = \phi$. Let $P = P_u + P_v$ and $Q = Q_u + Q_v$, where $P_u, Q_u \subseteq C_u$ and $P_v, Q_v \subseteq C_v$. We only consider the case when, $(P,Q) \neq (C_u, C_v)$. Therefore if $P_u = \phi$ or $P_v = \phi$, then $Q_u \neq \phi$ and $Q_v \neq \phi$ and vice versa. Without loss of generality, let us assume P_u and $P_v \neq \phi$. We get,

$$\begin{aligned} c(P,Q) &= c(P_u + P_v, Q_u + Q_v) \\ &= c(P_u, Q_u) + c(P_u, Q_v) \\ &\quad + c(P_v, Q_u) + c(P_v, Q_v) \\ &\geq c(P_u, Q_u) + c(P_v, Q_v) \\ &\geq c(x, Q_u) + c(x, Q_v) \quad \text{,By Lemma 1} \\ &\geq \alpha|Q_u| + \alpha|Q_v| \quad \text{,By construction} \\ &\geq \alpha|Q| \geq \alpha\, min\{|P|, |Q|\} \end{aligned}$$

Hence the proof of the lemma follows. □

CONDITION 3. To prove the claim of our algorithm under CONDITION 3, we first state an important lemma form [6]. This is obtained directly by the way min-cut tree is produced in [8].

Lemma 3 *Let T be the (unique) min-cut tree of an undirected graph G, and let A be a subtree of T. Let G' be the graph that results after contracting A in G, and let T' be the min-cut tree of G'. Let T'' be the tree that results after contracting A in T. Then T' and T'' are identical.*

The following lemma 4 is derived using Lemma 3.

Lemma 4 *Let C_u and C_v be two connected components obtained, after removing the artificial sink t from the created min-cut tree T of G. If there are some insertion and deletion of edges across and within the clusters C_u and C_v, then except C_u and C_v, all other clusters remain unaffected.*

Proof. Let G be the original graph and let us denote the graph after edge insertion and deletion as G'. Contract C_u and C_v in G and G', and call the contracted graphs H and H' respectively. Since all the edge insertions and deletions are within C_u and C_v, $H = H'$. Min-cut tree of H and H' are therefore identical and is same as the min-cut tree T of G after contracting C_u and C_v in T (by Lemma 3). Since contraction of C_u and C_v in T, does not affect the other communities, the proof follows. □

Observe that, adding t in G, as in basic "Cut-Clustering algorithm", and contracting $V - S$, has the same effect as contracting $V - S$ first in x and then adding an edge (t, x), of weight $\alpha|V_S|$ in the contracted graph. Also in the contracted graph, the contracted clusters, x, form a singleton cluster (follows directly from Lemma 4). With these observations, the claim of the algorithm under CONDITION 3 now follows from Lemma 4 and the correctness proof of the "Cut-Clustering algorithm" of [6].

2. Edge Deletion. The proofs of maintenance of clustering quality over intra-cluster and inter-cluster edge deletion are similar to inter-cluster and intra-cluster edge addition, respectively and are omitted for brevity.

3-4. Vertex Addition and Deletion. The addition and deletion of vertices can be viewed as a sequence of edge additions and deletions. Thereby the quality guarantee of the clusters over these update operations follows from the analysis of the previous cases.

3.5 Heuristic to Improve Time Complexity

Consider our incremental algorithm for inter-cluster edge insertion or intra-cluster edge deletion. For these cases, we need to compute minimum cut tree over a coarsened graph of size $O(k)$. This is equivalent to computing $O(k)$ maximum flow computations. To reduce the number of maximum flow computations, we use a heuristic, slightly modified from the one used in [6]. The heuristic of [6] is based upon the fact, that if the minimum cuts produced using vertices v_1 and v_2 as sources are $C_1, V/C_1$ and $C_2, V/C_2$, respectively, where $v_1 \in C_1$ and $v_2 \in C_2$, then either C_1 and C_2 are disjoint or one is a subset of another. On view of this, [6] sorts vertices in ascending order of the total wieght of the adjacent edges. Maximum flows for the vertices (as source) are computed according to the sorted order. However since maximum flow is same as minimum cut, while computing flow from vertex v_1, if v_2 belongs to the same cut as v_1, v_2 is marked to be in the same cluster as v_1. Maximum flow from v_2 is never computed. In practice this reduces the maximum flow computation to the number of clusters produced.

In our algorithm, we want to recluster one (for intra-cluster edge deletion) or two (for inter-cluster edge addition) clusters, at a time. Using this heuristic, the number of maximum flow computation will be reduced to $O(1)$. However we use a slight variation of the above heuristic. For each present cluster, C_i, $1 \leq i \leq s$, we mark one vertex, x_i, inside the cluster to be the prime vertex. We first compute the maximum flow from the prime vertex of any cluster. If the cut corresponding to the prime vertex does not include all the vertices, we then follow the above heuristic of [6].

If the cluster is a singleton, then the isolated vertex is marked as prime. Otherwise while updating the clusters for inter-cluster edge addition, under CONDITION 3, or intra-cluster edge deletion, we compute the minimum-cut tree over the coarsened graph. For each cluster over the coarsened graph, there exists a vertex, whose community covers the entire cluster. These vertices are marked as prime for each of these clusters, that contain vertices (uncontracted) of the original graph.

4 Experimental Results

In this section, we present the results of a preliminary experimental study on our incremental clustering algorithm. The results clearly demonstrate the superiority of our algorithm in terms of cluster quality and computation time. We

Graph	# vertex	# edge	Description
DATA	2851	15093	finite element mesh
3ELT	4720	13722	finite element mesh
UK	4824	6837	finite element mesh
ADD32	4960	9462	32 bit Adder
WHILAKER3	9800	28989	finite element mesh
CRACK	10240	30380	finite element mesh
FE-4ELT2	11143	32818	finite element mesh
MEMPLUS	17758	54196	memory circuit
BCSSTK30	28294	1007284	stiffness matrix

Table 1. Banchmark Graphs for Testing

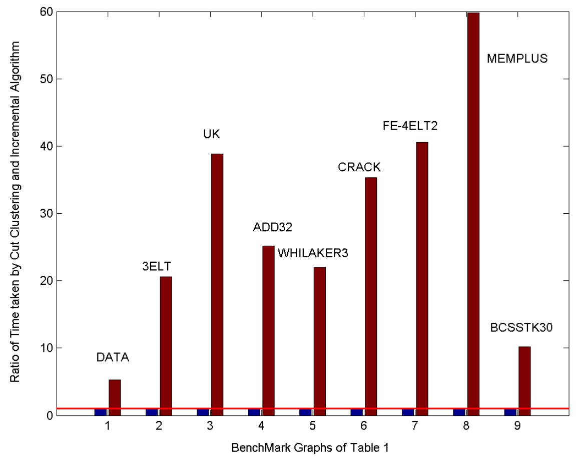

Figure 10. Computation Time of our Incremental Algorithm compared with *Cut Clustering Algorithm***. The Y-axis plots the ratio of the computation time of** *Cut Clustering Algorithm* **to that of our incremental algorithm. The bar above height** 1 **for** *Cut Clustering Algorithm* **indicates that our algorithm performs better.**

present comparison analysis of our algorithm with the *Cut Clustering Algorithm* (Section 2) [6] and the recent multi-level algorithm of [5] (MLKM), which has been shown to outperform cutting edge spectral algorithms [16] as well as METIS [11]. To speed up the computation of our incremental algorithm, we use the heuristics described in Section 3.5.

4.1 Data Set

We use 9 test graphs [2], listed in Table 1, as inputs in our experiments. They are obtained from various application domains and have been used as benchmark for testing MLKM algorithm [5] and METIS [11]. From each of these graphs, we create an edge-stream, by choosing edges randomly. The stream of edges is then presented sequentially to our incremental algorithm. Operations involving inter(intra)-cluster edge-deletion is similar to intra(inter)-cluster edge insertion. Hence testing our algorithm for insert-only edge stream suffices to prove its effectiveness.

4.2 Results

1. Comparison Results with ”Cut Clustering Algorithm”.

We compare the running time of our incremental algorithm with the *Cut Clustering Algorithm* (Section 2), using the benchmark graphs (Table 1). Both the algorithms are run 10 times, using different values of α and average computation time over these runs are used for comparison. Figure 10 shows the relative performance of the *Cut Clustering Algorithm* with our incremental algorithm. We plot the ratio of computation time taken by the *Cut Clustering Algorithm* to that of the incremental algorithm. For all the benchmark graphs taken, this ratio is much higher than 1. This validates the fast running time of our algorithm. It also shows that the advantage is much more for sprase graphs, than the dense ones.

The advantage is highest for MEMPLUS, on which our algorithm performs 60 times faster than the *Cut Clustering Algorithm*. For most of the graphs, the speed advantage, is more than 20 times. The advantage is the least for BCSTTK30 and DATA which are well dense. They run respectively 8 and 6 times faster than the *Cut Clustering Algorithm*. For same value of α, the clustering quality is nearly identical for both the algorithms and hence, we do not show comparison plot for clustering quality of these two algorithms.

2. Comparison Results with the Fast Multilevel Algorithm [5].

We compare the computation time as well as clustering quality produced by the fast multi-level algorithm (MLKM) of [5][3] with our incremental algorithm.

We use relative performance measure for our comparison analysis. That is, for comparing running time, we plot the ratio of the computation time taken by the MLKM and our incremental algorithm. For comparing clustering quality, we similarly take the ratio of values obtained, by the two algorithms, using ratio-association and normalized cut

[2]http://staffweb.cms.gre.ac.uk/˜ cwalshaw/partition

[3]Code downloaded from http://www.cs.utexas.edu/users/dml/Software

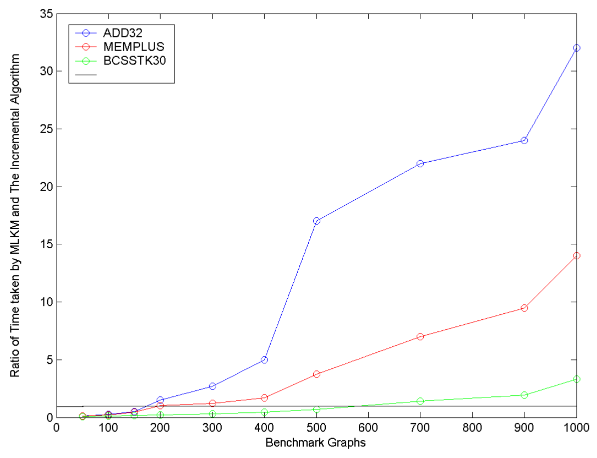

Figure 11. Computation Time of our Incremental Algorithm compared with MLKM algorithm. The Y-axis plots the ratio of the computation time of MLKM to that of our incremental algorithm. The X-axis plots the number of clusters. The value above 1 for MLKM indicates that our algorithm performs better.

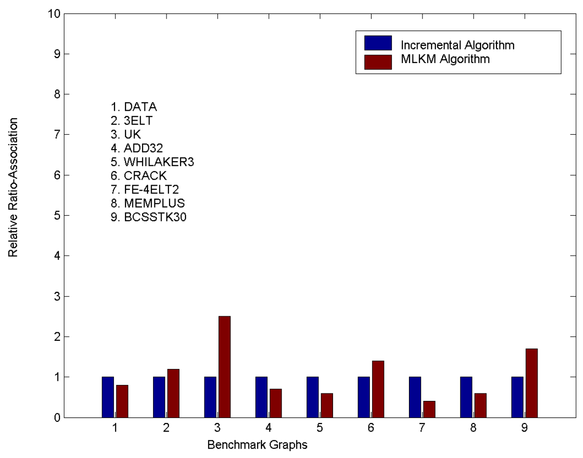

Figure 12. Ratio Association Value of our Incremental Algorithm compared with MLKM algorithm. The Y-axis plots the ratio of the ratio association value of MLKM to that of our incremental algorithm for 300 (nearly) clusters. Note that bars below height 1 for MLKM correspond to the cases where our algorithm performs better.

as measures (defined in [5]). Figure 11 presents the plot of relative computation time measured versus number of clusters, for three benchmark graphs-ADD32, MEMPLUS and BCSSTK30. It is clearly visible from the plot, that with increase in number of clusters, our incremental algorithm starts performing better than the MLKM algorithm of [5]. For BCSSTK30, which has more than 28 thousand vertices and 10 million edges, when the number of clusters becomes more than 630, our algorithm outperforms MLKM. For ADD32 and MEMPLUS, when the number of clusters produced is more than 150 and 250 respectively, our algorithm performs better. For comparing clustering quality, we use the measure of ratio-association value (RAV) and normalized-cut value (NCV) defined in [5]. They are closely related to the bicriteria, mentioned in Subsection 2.3. The goal is to obtain clusters that maximize ratio-association and minimize normalized cut. Figure 12 and 13 show the plot of relative RAV and NCV values for the benchmark graphs. The number of clusters taken is approximately 300. The plots indicate that the quality of clusters produced are nearly identical for both of the algorithms. However for 60% of the cases our algorithm performs marginally better than MLKM.

For ADD32, when the number of clusters is below 150, RAV is less than 500 for MLKM. This indicates at this stage the clusters are not well-formed. If the number of clusters is raised to 400, RAV crosses 1000 and still NCV is less than 100. This is an ideal situation for obtaining clusters. At this stage our incremental algorithm runs 5 times faster than MLKM. For MEMPLUS, which has more than 17 thousand vertices, RAV is 556, 720 and 897, when number of clusters is 150, 200 and 250, respectively. As the number of clusters grows to 300 and then to 500, RAV reaches 1059 and 1737 mark consequtively. However NCV value is only 95.5 and 169.9, respectively. These measures for MLKM indicates that, with number of clusters less than 300, clustering quality is not good. In this region, our algorithm performs again 4-5 times better than MLKM. Same applies for the other benchmark graphs. This shows the advantage of using our incremental algorithm over MLKM, clearly.

Therefore, our experiments demonstrate that when applied to stored graphs, our algorithm performs much better than both MLKM and *Cut Clusterin Algorithm*, in most of the practical scenarios. In addition to this, we know, our incremental algorithm is the only recourse when the edge relationship of the underlying graph evolves over time and clustering has to be performed efficiently in this dynamic environment. Neither MLKM nor *Cut Clustering Algorithm* can maintain clusters, without processing the entire graph afresh, when the underlying graph structure is changing.

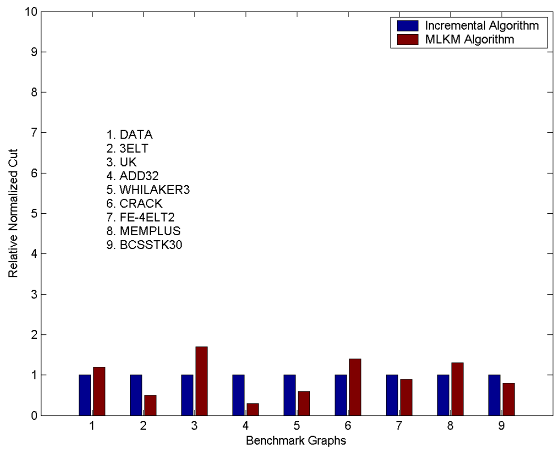

Figure 13. Normalized Cut Value of our Incremental Algorithm compared with MLKM algorithm. The Y-axis plots the ratio of the normalized cut value of MLKM to that of our incremental algorithm for 300 (nearly) clusters. Note that bars above height 1 for MLKM correspond to the cases where our algorithm performs better.

5 Conclusion

We present an incremental algorithm for graph clustering, using minimum cut tree. To the best of our knowledge, no other incremental clustering algorithm is known for graphs, using the bicriteria of [6]. Our algorithm efficiently maintains clusters over highly dynamic graphs, where edges and vertices are being added and removed constantly. We give detailed theoretical analysis of the quality of clusters obtained by our algorithm. The clustering quality is identical to that of [6]. However [6] requires reclustering of the entire graph, in face of edge(vertex) insertions and deletions. Even for stored graph, when edges are presented in a streaming fashion to our incremental algorithm, computation time for it is much better than that of [6]. This is verified by testing both the algorithms over 9 benchmark graphs, obtained from different applications.

Recently a multilevel kernel based algorithm has been proposed in [5]. The algorithm has been shown to outperform the cutting edge spectral algorithms, which has been the most popular tools for graph clustering over decades. Using the same benchmark graphs of ours, the algorithm in [5] has been showns to be much faster and producing better cluster quality than spectral algorithms. Our experimental results demonstrate that when number of clusters are not too low, our algorithm runs even faster than the algorithm in [5]. The clustering quality of both the methods are comparable, with our algorithm perfoming better in 60% of the benchmark graphs tested. Therefore our algorithm opens the possibility of obtaining high quality clusters over highly dynamic graphs and gives a better alternative for the fastest algorithm for graph clustering known so far, in most of the practical scenarios.

The time complexity of cluster maintenance over edge and vertex updates, depends superlinearly on the cluster size. Obtaining a graph clustering algorithm, that processes updates in time sublinear of the cluster size or independent of it, is still open.

References

[1] Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, 2004. Ossama Younis and Sonia Fahmy.

[2] Stefano Basagni. Distributed clustering for ad hoc networks. In *ISPAN ’99: Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN ’99)*, page 310, 1999.

[3] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA’03)*, volume 2832, pages 568–579, 2003.

[4] H. Chan and A. Perrig. ACE: An emergent algorithm for highly uniform cluster formation. In *Proceedings of the First European Workshop on Sensor Networks (EWSN’04)*, Jan 2004.

[5] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD*, pages 629–634, 2005.

[6] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees, Internet Mathematics, 1(3), 355-378, 2004.

[7] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD ’00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160, 2000.

[8] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J-SIAM*, 9(4):551–570, December 1961.

[9] Ekram Hossain, Rajesh Palit, and Parimala Thulasiraman. Clustering in mobile wireless ad hoc networks: issues and approaches. pages Wireless communications systems and networks, 383–424, 2004.

[10] R. Kannan, S. Vempala, and A. Veta. On clusterings-good, bad and spectral. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 367, 2000.

[11] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. Technical Report 95-035, University of Minnesota, june 1995.

[12] R. Kellogg, A. Heath, and L. Kavraki. Clustering Metabolic Networks Using Minimum Cut Trees, http://cohesion.rice.edu/engineering/Computer Science/emplibrary/AHeath.ppt , 2004.

[13] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[14] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL, Main Volume, 2004, pages 271-278, Barcelona*.

[15] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(11):1101–1113, 1993.

[16] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 313, 2003.