

# Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree \*

Barna Saha

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

Pabitra Mitra

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

## Abstract

We present an efficient dynamic algorithm for clustering undirected graphs, whose edge property is changing continuously. The algorithm maintains clusters of high quality in presence of insertion and deletion (update) of edges. The algorithm is motivated by the minimum-cut tree based partitioning algorithm of [3] and [4]. It takes  $O(k^3)$  time for each update processing, where  $k$  is the maximum size of any cluster. This is the worst case time complexity, and in general update time taken is much less. The clusters satisfy the bicriteria for quality guarantee proposed in [3].

## 1 Introduction

Graph Clustering, partitions a graph into several disjoint sub-graphs, such that each cluster is heavily connected, and inter-cluster connectivity is low. The problem has intrinsic relevance in computer science, mathematics and many applied areas. The most popular bicriteria measure for “good clustering” in this respect, is given by Kannan et al. [6]. A slight variant of the above criteria is proposed in [3]. In reality, both these measures perform well.

There exists large varieties of graph clustering algorithms, based on spectral clustering [6], rapid mixing of Markov chains [1], multilevel graph partitioning schemes like METIS [7] and MLKM algorithm of [2] etc.. MLKM algorithm is so far the fastest algorithm known for graph clustering. A new direction to graph-clustering has been introduced by modeling the clustering problem, using minimum-cut, maximum-flow problem of the underlying graph [4, 3]. Flake et al. has used this method to produce clusters [3], with theoretical quality measure, that works remarkably in practice [3] and has also been used as learning algorithm [8].

However if the underlying graph is dynamic in nature, then with each change in the edge relationship, these algorithms need to be run afresh on the entire graph. Many real world networks can be modeled as graphs and most of these graphs are dynamic and evolving continuously. One such example is the World Wide Web (WWW). 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 [4]. However the highly dynamic nature of the web renders all the existing 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. The other examples include the citation graph, graphs generated from ad hoc sensor networks, mobile networks etc.. Hence requirement for developing dynamic clustering algorithms which can handle changes in edge-relationships effectively and yet produces clusters of high quality comes naturally.

### 1.1 Contribution

- Our contributions are as follows:
- We develop a dynamic graph clustering algorithm that handles insertion and deletion of edges, while clustering is on progress, in  $O(k^3)$  time. 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. 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 [3], the time taken to perform clustering is  $O(n(\log n)^3)$ , in contrast to the  $O(n^3)$  algorithm of [3].
  - The quality of the clusters produced by our algorithm matches the quality requirements given in [3]. A detailed theoretical analysis to establish the clustering quality is provided.
  - The effectiveness of the algorithm is substantiated through experimentation on several benchmark graphs, upto 30 thousand nodes and 1 million edges. Comparison results with the algorithm of [3] and MLKM [2] clearly demonstrates the superiority of our dynamic algorithm.

**1.2 Organization** The rest of the paper is organized as follows. Section 2 delineates the clustering algorithm based on minimum-cut tree developed in [3]. Section 3 describes

\*Preliminary version of this work appeared in ICDM Workshops, 2006 [10]

our proposed dynamic graph clustering algorithm and gives detailed proof of its quality guarantee. Section 4 gives the experimental results and we conclude in section 5.

## 2 Clustering Using Mincut Tree

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 [5], 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.

**2.1 Clustering Algorithm** [3] defines clustering based on 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  $\alpha$  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 gives the basic *Cut Clustering Algorithm*.

---

*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 1: Cut Clustering Algorithm of [3]

**2.2 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.

[3] assures that if  $S$  is a cluster produced by the *Cut Clustering Algorithm*, then the following conditions are satisfied(See Fig 2):

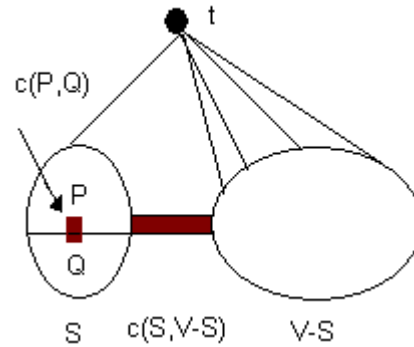


Figure 2: Inter and Intra Cluster Cuts

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  $\alpha$  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. [6], but closely similar to it and has its own advantage.

## 3 Dynamic Graph Clustering with Min-Cut Tree

In this section we present our main contribution—a dynamic algorithm for graph-clustering. The clustering technique is motivated by the *Cut Clustering Algorithm* described in the previous section.

Our proposed algorithm maintains for every vertex, two variables, *In Cluster Weight (ICW)* and *Out Cluster Weight (OCW)*. Let  $C_1, C_2, \dots, C_s$  ( $s > 0$  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 [2]. 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. 1). The algorithm supports and maintains clusters over insertion and deletion of edges and vertices. Edge insertion and deletion involve edges, where end vertices have already been seen.

Let  $\mathcal{C} = \{C_1, C_2, \dots, C_s\}$ , 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 describe our algorithm over these update operations.

### 1. Edge Insertion.

#### Intra-Cluster Edge Insertion.

---

```

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 3: Intra-Cluster Edge Addition

When an edge gets added, whose both end vertices belong to the same cluster, the cluster becomes more well-connected. Therefore, in this case, we simply update the adjacency matrix and *ICW*. The clusters remain unchanged.

Figure 3 gives the algorithm for *intra-cluster edge addition*. The update time is  $O(1)$ .

**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 2.2, 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 4: Merging of Two Clusters

*Merging of Clusters.* Merging of two clusters  $C_u$  and  $C_v$  is described in Figure 4. By merging  $C_u$  and  $C_v$ , 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  $\Theta(|C_u| + |C_v|) = \Theta(|C_u + C_v|)$ .

---

```

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 5: Creating Contracted Graph

*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,  $A$  represents a single or multiple clusters. The process to contract clusters is described in Figure 5. The contracted graph can be obtained from the adjacency matrix of the original graph along with  $ICW$  and  $OCW$  in time  $\Theta(|A|^2)$ .

---

```

Inter-Cluster Edge Addition $((i, j), w(i, j), \alpha)$ 
begin
Let  $i \in C_u$  and  $j \in C_v$ 
If  $\frac{\sum_{u \in C_u} OCW(u) + w(i, j)}{\sum_{v \in C_v} OCW(v) + w(i, j)} \leq \alpha$  and
 $\frac{\sum_{v \in C_v} OCW(v) + w(i, j)}{|V - C_u|} \leq \alpha$  {(CASE 1)}
Then
  Update  $A(i, j) + = w(i, j)$ 
  Update  $OCW(i) + = w(i, j)$ ,  $OCW(j) + = w(i, j)$ 
  Return  $C$ 
Else
If  $\frac{2c(C_u, C_v)}{V} \geq \alpha$  {(CASE 2)}
  Then
     $D = MERGE(C_u, C_v)$ 
  Return  $C + D - \{C_u, C_v\}$ 
Else {(CASE 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  $\{D_1, D_2, \dots, D_k\}$ ,  $k > 0$ , are the connected components of  $T''$  after removing  $t$ , containing vertices of  $C_u$  and  $C_v$ .
   $C = \{D_1, D_2, \dots, D_k, C_1, C_2, \dots, C_s\} - \{C_u, C_v\}$ 
  Return  $C$ 
end

```

---

Figure 6: Inter-Cluster Edge Addition

Now we are ready to describe our algorithm for *inter-cluster edge addition* (Figure 6). If the addition of edge does not deteriorate the clustering quality (CASE 1), then the same clusters are maintained. Else if, CASE 2 is satisfied, then the two clusters,  $C_u$  and  $C_v$ , containing the end-vertices of the inserted edge, are merged. Otherwise (CASE 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 the 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  $\alpha$ . 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  $C - \{C_u, C_v\}$  are returned as the clusters of the original graph. The entire process takes time  $\Theta(|C_u + C_v|^3)$ .

## 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 CASE 3. Here the end vertices of the deleted edge belong to the same cluster  $C_u$ . So except  $C_u$ , all the other clusters are contracted to form the coarsened graph. The time complexity to process the update is  $\Theta(|C_u|^3)$ , which can be reduced using a heuristic to  $\Theta(|C_u|^2)$ . The detailed algorithm is given in Figure 7.

---

```

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  $\{D_1, D_2, \dots, D_k\}$ ,  $k > 0$ , are the connected components of  $T''$  after removing  $t$ , containing vertices of  $C_u$ 
   $C = \{D_1, D_2, \dots, D_k, C_1, C_2, \dots, C_s\} - \{C_u\}$ 
  Return  $C$ 
End

```

---

Figure 7: Intra Cluster Edge Deletion

**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 8). The time taken by the process is  $O(1)$ .

---

```

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 8: Inter-Cluster Edge Deletion

**3. Vertex Addition and Deletion** An addition of a new vertex, with edges incident on it, may be viewed as creation of a new cluster containing the new vertex as an isolated node and then processing all the edges incident on it as in the case of “Edge Insertion”. Similarly vertex deletion can be handled, by first removing all the edges incident on that vertex, by the process of edge deletion and then deleting the isolated vertex.

**Time Complexity** Let  $k = \max_{u=1}^s \{|C_u|\}$ . We see time complexity is dominated by the operations of inter-cluster edge insertion and intra-cluster edge deletion. Therefore update-processing time is  $O(k^3)$ . When edges can only be added, or when the graph is stored in the secondary disk apriori, time required for clustering is  $O(mk^3)$  (actually much less than this). Generally the clusters are small. So if  $k = O(\log n)$ , and since most of the massive graphs that occur in real life have very low average degree [3], the time complexity becomes  $O(n \text{polylog}(n))$  compared to  $O(n^3)$  time requirement of the offline *Cut Clustering Algorithm*.

**Proof of Clustering Quality** In this section, we show that the clusters obtained by our dynamic algorithm, has the same quality guarantee (Subsection 2.2) of the offline clustering algorithm of [3].

**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  $\alpha$ . 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.**

*CASE 1.* Note that,  $\sum_{u \in C_u} OCW(u) = C(C_u, V-C_u)$ .

Therefore if CASE 1 is satisfied, by quality guarantee of the existing clusters, both the criterias of Subsection 2.2 are satisfied.

*CASE 2.* Lemma 3.1 is obtained from the property of the minimum-cut tree. Lemma 3.2 establishes the quality guarantee of our algorithm under CASE 2, using Lemma 3.1.

**LEMMA 3.1.** *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 [3]. □

**LEMMA 3.2.** *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, by direct calculation  $c(D, V-D) = \alpha(|V-D|) + \alpha|V| - 2c(C_u, C_v)$ . If  $\alpha \leq \frac{2c(C_u, C_v)}{|V|}$ , 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,

$$\begin{aligned}
c(P, Q) &= c(P_u + P_v, Q_u + Q_v) \\
&\geq c(P_u, Q_u) + c(P_v, Q_v) \\
&\geq c(x, Q_u) + c(x, Q_v) \text{ ,By Lemma 3.1} \\
&\geq \alpha|Q_u| + \alpha|Q_v| \text{ ,By construction} \\
&\geq \alpha|Q| \geq \alpha \min\{|P|, |Q|\}
\end{aligned}$$

□

*CASE 3.* To prove the claim of our algorithm under CASE 3, we first state an important lemma, Lemma 3.3, form [3]. This is obtained directly by the way min-cut tree is produced in [5].

**LEMMA 3.3.** *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 3.4 is derived using Lemma 3.3.

**LEMMA 3.4.** *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 3.3). Since contraction of  $C_u$  and  $C_v$  in  $T$ , does not affect the other communities, the proof follows.  $\square$

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. In the contracted graph, the contracted clusters,  $x$ , form a singleton cluster (follows directly from Lemma 3.4). With these observations, the claim of the algorithm under CASE 3 now follows from Lemma 3.4 and the correctness proof of the *Cut Clustering Algorithm* of [3].

The proofs of maintenance of clustering quality over edge deletions and vertex insertions and deletions are similar and omitted for brevity.

**Heuristic to Improve Time Complexity** A Heuristic may be applied to improve the time complexity of each update to  $O(k^2)$ . It employs marking a special vertex of each cluster as “prime” and computing the first minimum cut with the prime vertex as source (of the cluster involved), while computing the partial minimum cut tree. After that the heuristic of [3] is followed. The details of this may be found in [9].

## 4 Experimental Results

The results of a preliminary experimental study on our dynamic clustering algorithm demonstrate the superiority of our algorithm in terms of cluster quality and computation time over the *Cut Clustering Algorithm* (Section 2) [3] and the recent multi-level algorithm (MLKM) [2].

We use the test graphs <sup>1</sup>, 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 [2] and METIS [7]. From each of these graphs, we randomly choose edges one by one and present it as the recent update to the algorithms under comparison. The detailed results of the experiments and comparison analysis may be found in [9] and are not given here for lack of space.

## 5 Conclusion

We present a dynamic algorithm for graph clustering, using minimum cut tree. To the best of our knowledge, no other dynamic clustering algorithm is known for graphs, using the bicriteria of [3]. For evolving graphs, our algorithm runs much faster than the *Cut Clustering Algorithm* [3] and the

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: Benchmark Graphs for Testing

recent multi-level algorithm (MLKM) [2]. The clustering quality is identical to that of *Cut Clustering Algorithm* [3] and comparable to MLKM. The time complexity of cluster maintenance over updates, depends superlinearly on the cluster size. Reducing update time to sublinear of the cluster size or independent of it, is still open.

## References

- [1] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering. In *ESA'03*, volume 2832, pages 568–579, 2003.
- [2] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD'05*, pages 629–634, 2005.
- [3] G. W. Flake, R. E. Tarjan, and K. Tsioutsoulouklis. Graph clustering and minimum cut trees, *Internet Mathematics*, 1(3), 355-378, 2004.
- [4] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD '00:*, pages 150–160, 2000.
- [5] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J-SIAM*, 9(4):551–570, December 1961.
- [6] R. Kannan, S. Vempala, and A. Veta. On clusterings-good, bad and spectral. In *FOCS '00:*, page 367, 2000.
- [7] 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.
- [8] 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*.
- [9] B. Saha and P. Mitra. Dynamic Algorithm for Graph Clustering, <http://home.iitk.ac.in/~barna/dc.pdf>, July 2006.
- [10] Barna Saha and Pabitra Mitra. Dynamic algorithm for graph clustering using minimum cut tree. In *ICDM Workshops*, pages 667–671, 2006.

<sup>1</sup><http://staffweb.cms.gre.ac.uk/~cwalshaw/partition>