Scalable and Memory-Efficient Clustering of Large-Scale Social Networks

Joyce Jiyoung Whang
Department of Computer Science
The University of Texas at Austin
Austin, USA
joyce@cs.utexas.edu

Xin Sui
Department of Computer Science
The University of Texas at Austin
Austin, USA
xinsui@cs.utexas.edu

Inderjit S. Dhillon
Department of Computer Science
The University of Texas at Austin
Austin, USA
inderjit@cs.utexas.edu

***Abstract*—Clustering of social networks is an important task for their analysis; however, most existing algorithms do not scale to the massive size of today's social networks. A popular class of graph clustering algorithms for large-scale networks, such as PMetis, KMetis and Graclus, is based on a multilevel framework. Generally, these multilevel algorithms work reasonably well on networks with a few million vertices. However, when the network size increases to the scale of 10 million vertices or greater, the performance of these algorithms rapidly degrades. Furthermore, an inherent property of social networks, the power law degree distribution, makes these algorithms infeasible to apply to large-scale social networks. In this paper, we propose a scalable and memory-efficient clustering algorithm for large-scale social networks. We name our algorithm GEM, by mixing two key concepts of the algorithm, Graph Extraction and weighted kernel *k*-Means. GEM efficiently extracts a good skeleton graph from the original graph, and propagates the clustering result of the extracted graph to the rest of the network. Experimental results show that GEM produces clusters of quality comparable to or better than existing state-of-the-art graph clustering algorithms, while it is much faster and consumes much less memory. Furthermore, the parallel implementation of GEM, called PGEM, not only produces higher quality of clusters but also achieves much better scalability than most current parallel graph clustering algorithms.**

***Keywords*-clustering; social networks; graph clustering; scalable computing; graph partitioning; kernel *k*-means;**

I. Introduction

Social network analysis has received significant attention in recent years. Social networks can be modeled as graphs with vertices and edges where vertices indicate individual actors, and edges indicate social relationships between the actors. The resulting graphs are large and complex.

Graph clustering, also known as graph partitioning, is one of the most fundamental and important techniques for analyzing the structure of a network. Graph clustering algorithms partition a graph so that closely connected vertices are assigned to the same cluster. In the social network analysis context, each cluster can be considered as a community in the network.

Due to the massive size of real networks, scalability is a critical issue. The most commonly used graph clustering algorithms, which are known to be fast and scalable, are based on a multilevel framework, for example, PMetis [1], KMetis [2], and Graclus [3]. Generally, multilevel-based algorithms work reasonably well on networks with a few million vertices. However, when the network size increases to the scale of 10 million vertices or greater, the performance of multilevel-based algorithms can rapidly degrade. Specifically, when the degree distribution of a large network follows a power law, as is common for social networks, the degradation is quite significant. Therefore, existing multilevel-based graph clustering algorithms are not a good choice for clustering large-scale social networks. In our experiments, for example, KMetis takes about 19 hours to cluster a Twitter graph which contains about 50 million vertices and one billion edges, while consuming more than 180 Gigabytes memory.

In this paper, we propose a scalable and memory-efficient clustering algorithm for large-scale social networks. We name our algorithm GEM, by mixing two key concepts of the algorithm, Graph Extraction and weighted kernel *k*-Means. The main idea of GEM is to extract a good skeleton of the original graph, cluster the skeleton graph using weighted kernel *k*-means, and propagate the clustering result to the original graph.

It is generally known that social networks have a hierarchical structure. This structure allows us to extract a representative graph, which can be small but captures the overall structure of the original network well. Intuitively, clustering the representative subgraph can be beneficial to clustering the original graph. GEM extracts a subgraph using high degree vertices in the network. In the social network context, high degree vertices correspond to popular or influential people in the network. After clustering the representative subgraph, we propagate the clustering result to the rest of the network.

Experimental results show that GEM produces clusters of quality comparable to or better than existing state-of-the-art graph clustering algorithms while it is much faster and consumes much less memory. We also parallelize our algorithm — our parallel implementation, called PGEM, scales well with the number of processes. Compared to ParMetis [4], which is a popular parallel graph clustering library, PGEM not only produces higher quality of clusters but also achieves much better scalability.

II. Preliminaries

In this section, we formally state the graph clustering problem, and introduce the standard weighted kernel k-means algorithm which has been shown to be effective in optimizing graph clustering objectives.

A. Graph Clustering

A graph $G = (\mathcal{V}, \mathcal{E})$ is defined by a set of vertices $\mathcal{V}$, and a set of edges $\mathcal{E}$. Given a graph G, we can construct the corresponding adjacency matrix A such that $A_{ij} = e_{ij}$ if there is an edge between vertex i and vertex j, and 0 otherwise, where e_{ij} denotes an edge weight between vertex i and vertex j. Formally, given a graph $G = (\mathcal{V}, \mathcal{E})$, graph clustering seeks to partition the graph into k disjoint clusters $\mathcal{V}_1, \cdots, \mathcal{V}_k$ such that $\mathcal{V} = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_k$. Many different kinds of graph clustering objectives have been proposed and studied. We focus on two most popular objectives: Kernighan-Lin objective which is used in PMetis and KMetis, and the normalized cut objective which is used in Graclus.

Let us define $links(\mathcal{V}_p, \mathcal{V}_q)$ to be the sum of edge weights between two vertex sets $\mathcal{V}_p$ and $\mathcal{V}_q$. That is,

$$links(\mathcal{V}_p, \mathcal{V}_q) = \sum_{i \in \mathcal{V}_p, j \in \mathcal{V}_q} A_{ij}.$$

Let us also define the degree of a cluster to be the sum of edge weights of the vertices in the cluster, i.e., $degree(\mathcal{V}_p) = links(\mathcal{V}_p, \mathcal{V})$.

Kernighan-Lin objective aims to partition a graph into k equal-sized clusters while minimizing the cut between clusters. We can represent this objective as follows:

$$\min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{i=1}^{k} \frac{links(\mathcal{V}_i, \mathcal{V} \backslash \mathcal{V}_i)}{|\mathcal{V}_i|} \text{ such that } |\mathcal{V}_i| = \frac{|\mathcal{V}|}{k}. \quad (1)$$

On the other hand, the normalized cut objective aims to minimize the cut between clusters relative to the degree of a cluster. This objective is represented as follows:

$$\min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{i=1}^{k} \frac{links(\mathcal{V}_i, \mathcal{V} \backslash \mathcal{V}_i)}{degree(\mathcal{V}_i)}. \quad (2)$$

B. Weighted Kernel k-Means

It has been shown that a general weighted kernel k-means objective is equivalent to a weighted graph clustering objective [3]. This equivalence allows us to optimize a weighted graph clustering objective by using a weighted kernel k-means algorithm, which we now explain. Given a set of data vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$, the weighted kernel k-means algorithm aims to find k clusters $\pi_1, \pi_2, ..., \pi_k$ that minimize the objective J which is defined as follows:

$$J = \sum_{c=1}^{k} J_c, \text{ where } J_c = \sum_{\mathbf{x}_i \in \pi_c} w_i ||\phi(\mathbf{x}_i) - \mathbf{m}_c||^2 \quad (3)$$

where w_i is a nonnegative weight, ϕ is a non-linear mapping, and $\mathbf{m}_c$ is the centroid of the c-th cluster which is defined by:

$$\mathbf{m}_c = \frac{\sum_{\mathbf{x}_i \in \pi_c} w_i \phi(\mathbf{x}_i)}{\sum_{\mathbf{x}_i \in \pi_c} w_i}. \quad (4)$$

Given the kernel matrix K, where $K_{ij} = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$, we can compute the squared distance between $\phi(\mathbf{x}_i)$ and the cluster centroid $\mathbf{m}_c$, denoted by $||\phi(\mathbf{x}_i) - \mathbf{m}_c||^2$, as follows:

$$K_{ii} - \frac{2\sum_{\mathbf{x}_j \in \pi_c} w_j K_{ij}}{\sum_{\mathbf{x}_j \in \pi_c} w_j} + \frac{\sum_{\mathbf{x}_j, \mathbf{x}_l \in \pi_c} w_j w_l K_{jl}}{(\sum_{\mathbf{x}_j \in \pi_c} w_j)^2}. \quad (5)$$

The weighted kernel k-means algorithm computes the closest centroid for every data point, and assigns the data point to its closest cluster. After all the data points are considered, the centroids of clusters are updated. This procedure is repeated until the change in the objective value becomes sufficiently small or a maximum number of iterations is reached.

Now, let us return to the graph clustering problem. Recall that the normalized cut objective is defined in (2). This objective has been shown to be equivalent to the weighted kernel k-means objective by defining the weights and the kernel matrix as follows: we define the weight of each data point to be the degree of each vertex, and the kernel matrix to be $K = \sigma D^{-1} + D^{-1} A D^{-1}$, where D is a degree diagonal matrix with $D_{ii} = \sum_{j=1}^{n} A_{ij}$, and σ is a scalar that makes the kernel positive-definite [3]. Then, we can represent (5) using only graph-theoretic terms. That is, we can quantify the squared distance between a vertex $\hat{v}$ and the cluster $\mathcal{V}_c$ as follows:

$$dist(\hat{v}, \mathcal{V}_c) = \begin{cases} \hat{\alpha} - \dfrac{\sigma}{degree(\mathcal{V}_c)} & \text{if } \hat{v} \in \mathcal{V}_c \\ \hat{\alpha} + \dfrac{\sigma}{degree(\mathcal{V}_c)} & \text{if } \hat{v} \notin \mathcal{V}_c; \end{cases} \quad (6)$$

$$\text{where } \hat{\alpha} = \frac{\sigma}{\hat{d}} - \frac{2links(\hat{v}, \mathcal{V}_c)}{\hat{d} \cdot degree(\mathcal{V}_c)} + \frac{links(\mathcal{V}_c, \mathcal{V}_c)}{degree(\mathcal{V}_c)^2}.$$

where $\hat{d}$ denotes the degree of $\hat{v}$, and $links(\hat{v}, \mathcal{V}_c)$ denotes the sum of edge weights between $\hat{v}$ and the vertices in $\mathcal{V}_c$. Using this distance metric, we can assign every vertex to its closest cluster using the weighted kernel k-means algorithm.

III. Motivation

We introduce the multilevel framework for graph clustering, which has been employed for popular graph clustering algorithms [1], [2], [3]. We analyze this framework, and state their limitations on large-scale social networks.

A. Multilevel Framework for Graph Clustering

The multilevel framework has been known to be an efficient way to solve large-scale graph clustering problem. A popular class of graph clustering algorithms, such as PMetis, KMetis, and Graclus are based on the multilevel framework. Specifically, PMetis optimizes Kernighan-Lin objective using multilevel recursive bisection, and KMetis adopts a multilevel k-way refinement algorithm. Graclus optimizes the normalized cut objective by multilevel refinement using the weighted kernel k-means algorithm.

The multilevel framework consists of three phases: coarsening phase, initial clustering phase, and refinement phase.

1) Coarsening Phase: We repeatedly transform the original graph into smaller graphs, level by level. Starting with the original graph G_0, we generate a series of smaller graphs $G_1, G_2, \cdots, G_l$ such that $|\mathcal{V}_i| > |\mathcal{V}_{i+1}|$ for all $i = 0, \cdots, l-1$. Given G_i, we construct G_{i+1} as follows: At the beginning, all vertices are unmarked. We randomly visit a vertex, and merge this vertex with one of its unmarked neighbors. When we merge two vertices, the edge between them is hidden, and the resulting vertex is considered as a supernode which has all the edges the two vertices had. When a vertex is visited or merged with another vertex, we mark it. If a vertex has no unmarked neighbors, we just leave it and mark it. We repeat this procedure until all vertices are marked or no further coarsening can be performed.

2) Initial Clustering Phase: Once we get a small enough graph by multilevel coarsening, we can directly partition the coarsest graph. The initial clustering algorithm used at this stage varies depending on PMetis, KMetis, and Graclus.

3) Refinement Phase: The refinement phase proceeds in a reverse direction of the coarsening phase. Now, we are given a clustering of G_{i+1}, and derive a clustering of G_i for all $i = l-1, \cdots, 0$. First, we project a clustering of G_{i+1} to G_i. Then, we refine the clustering of G_i using a refinement algorithm which also varies depending on PMetis, KMetis, and Graclus. By repeatedly refining the clustering level by level, we get the final clustering of the original graph.

B. Limitations of Multilevel Framework on Large-Scale Social Networks

We analyze the multilevel framework, and state its limitations.

1) Difficulties of Coarsening in Large Networks: In the coarsening from G_i to G_{i+1}, the graph reduction ratio can be defined by:

$$\text{Graph Reduction Ratio} = \frac{|\mathcal{V}_i|}{|\mathcal{V}_{i+1}|}.$$

Recall that we merge up to two vertices to form a supernode. By definition, the graph reduction ratio cannot be smaller than 1, and cannot be bigger than 2. Clearly, a higher reduction ratio indicates a more effective coarsening procedure.

The success of the multilevel framework algorithms is based on the assumption that the graph reduction ratio in the coarsening step is reasonably high at each level. However, for real large social networks, this turns out not to be the case. Specifically, when a network follows power law degree distribution, the graph reduction ratio becomes much smaller. This is because of the innate structure of the network. When the degree distribution follows a power law, most of the low degree vertices in the network tend to be attached to high degree vertices. Once these high degree vertices are all merged with some low degree vertices, the rest of the low degree vertices have little chance to be merged with any vertex. As a result, there remain a large number of vertices which cannot be merged with any vertex, at each level. We observe that the graph reduction ratio is pretty small on real large social networks. For example, Figure 1 (a) shows the graph reduction ratio of Metis on our Twitter network which is presented in Section VI. We see that the ratio is much smaller than 2 across all the levels, which indicates that the coarsening procedure is ineffective.

Figure 1 (b) shows the number of edges of the coarsened graph at each level. A good coarsening step should hide a large number of edges from one level to the next level [5], which means that the difference between the number of edges of consecutive levels should be large. For the Twitter network, however, the number of edges is not significantly reduced at each level.

2) Memory Consumption: Another important issue is memory consumption. Since the multilevel framework generates a series of graphs in the coarsening phase, it requires quite a bit of extra memory for storing the graphs. If the size of the original graph is very large, however, sometimes it is infeasible to allocate such extra memory. Figure (c) shows the cumulative memory consumption in the coarsening phase of the Twitter network. When we store a large network, we usually use an adjacency list data structure. In this case, the minimum space complexity of storing a graph is $O(|\mathcal{V}| + |\mathcal{E}|)$. Since the size of the Twitter network is not significantly reduced at each level, the total memory consumption increases rapidly.

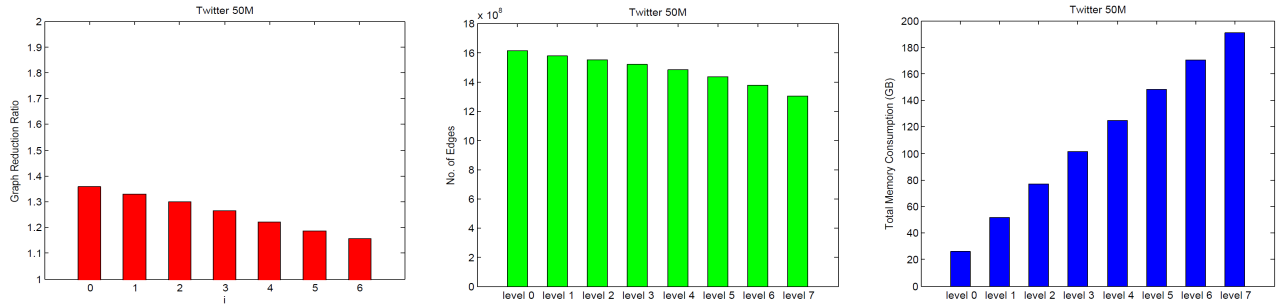
3) Difficulties in Parallelization: Graph clustering algorithms are widely recognized to be hard for parallelization [6]. The multilevel framework based graph clustering algorithms have presented significant obstacles to large-scale parallelization. First, graph coarsening is done by edge contractions, where each vertex is only allowed to participate in one edge contraction. To ensure this property, a global consensus has to be reached among processes, which requires intensive communication between processes. In addition, the consensus may become more and more difficult to achieve as the coarsening proceeds to deeper levels, since coarsening tends to make graphs denser. This further increases communication between processes.

IV. Scalable Graph Clustering using Graph Extraction and Kernel k-Means

We propose GEM, a scalable and memory-efficient graph clustering algorithm for large-scale social networks. GEM consists of three phases. First, we extract a representative subgraph of the original graph. Then, we cluster this subgraph which is much smaller than the original graph. Once we obtain clustering of this small graph, we propagate it to the rest of the network. Finally, we refine the clustering of the original graph.

A. Graph Extraction

In GEM, we extract a skeleton of the original graph by choosing high degree vertices in the graph. We select vertices whose degrees are higher than a certain threshold. We can specify the degree threshold such that a desired number



(a) Graph Reduction Ratio (b) No. of Edges at Each Level (c) Total Memory Consumption

Figure 1. Metis Coarsening Phase of Twitter 50M Network — existing multilevel algorithms perform poorly. (a) The graph reduction ratio from one level to the next level is much smaller than 2, which indicates that the coarsening step is ineffective. (b) Coarsening step fails to reduce the number of edges in the graph. (c) The total memory consumption increases rapidly during coarsening phase.

of extracted vertices is achieved. Then, we construct the subgraph of the original graph by extracting the vertices and edges between the selected vertices. This induced subgraph is considered a skeleton of the original graph.

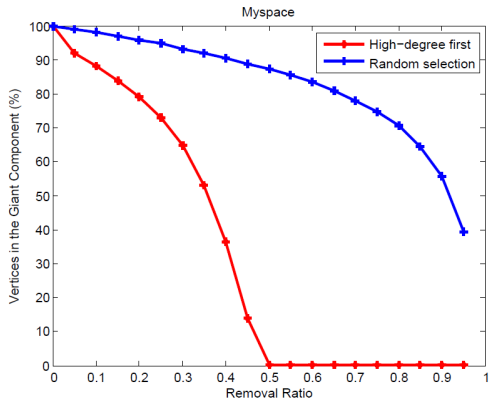


Figure 2. Vertices in the giant component of remaining vertices — high degree vertices play a pivotal role in preserving the structure of a network. The network collapses rapidly when high degree vertices are removed first.

Power law degree distribution of social networks indicates that a small set of high degree vertices covers a large portion of the edges in the network. Figure 2 shows the importance of high degree vertices in maintaining connectivity of a network. Detailed information about the network is presented in section VI. To see the role of high degree vertices in preserving the structure of a network, we remove a certain number of vertices in the network according to their degrees. After removing the vertices, we compute the giant connected component among the remaining vertices, and measure how many vertices are included in the giant connected component. If the giant component contains a substantial portion of the remaining vertices, then it indicates that the remaining vertices are well connected to each other. As can be seen in Figure 2, the network collapses rapidly when high degree vertices are removed first. Compared to the case where we randomly remove vertices, the remaining vertices become more disconnected to each other when we remove high degree vertices first. It is clear that these high degree vertices play a pivotal role in maintaining the overall structure of the network. This property of high degree vertices is aligned with the properties that a good skeleton of a graph must satisfy.

It is well known that social networks exhibit a hierarchical structure. In a social network, high degree vertices correspond to popular and influential people. Intuitively, high degree vertices have more chance to occupy higher levels in the hierarchy. Therefore, a subgraph which consists of high degree vertices can be considered a good skeleton of the original graph. In this sense, clustering the subgraph can be beneficial to clustering the original graph.

B. *Clustering of Extracted Graph*

Once a skeleton graph is extracted from the original graph, the extracted graph is clustered using a weighted kernel k-means algorithm. Specifically, GEM optimizes the normalized cut using a weighted kernel k-means algorithm.

It is generally known that k-means algorithm has the following weaknesses: (i) the final clustering result tends to be significantly affected by initialization of clusters, and (ii) empty clusters can be accidentally generated. In this subsection, we explain how we resolve these problems.

1) Finding Seeds: We refer to initial data points in clusters as *seeds*. A general idea of finding good seeds is to place seeds in a way that they are as far as possible from each other. Note that in the weighted kernel k-means setting, the squared distance between two data points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ is represented by:

$$||\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)||^2 = K_{ii} - 2K_{ij} + K_{jj}. \tag{7}$$

Recall that when we optimize the normalized cut objective using a weighted k-means algorithm, we set the kernel matrix as $K = \sigma D^{-1} + D^{-1}AD^{-1}$. In this setting, we can represent (7) using only graph-theoretic terms. The distance between two vertices $\phi(v_i)$ and $\phi(v_j)$, denoted by $||\phi(v_i) - \phi(v_j)||^2$, is as follows:

$$\begin{cases} \dfrac{\sigma}{d_i} - \dfrac{2e_{ij}}{d_i d_j} + \dfrac{\sigma}{d_j} & \text{if } (v_i, v_j) \in \mathcal{E} \\ \dfrac{\sigma}{d_i} + \dfrac{\sigma}{d_j} & \text{if } (v_i, v_j) \notin \mathcal{E} \end{cases} \tag{8}$$

where d_i denotes the degree of vertex i, and e_{ij} denotes an edge weight between v_i and v_j. We can see that low degree vertices are far away from each other, which indicates that those vertices would be good candidates for seeds. However, sorting the vertices according to their degrees can be quite expensive in large graphs.

We develop a fast *seeding* strategy which is empirically shown to be effective in clustering the extracted graph. We call our seeding strategy *Down-Path Walk Algorithm*.

We refer to a path $v_i \rightarrow v_j$ as a *down-path* if $d_i \geq d_j$. The concept of a down-path was first introduced in [7] where the hierarchical topology of a network is quantified. Integrating this concept with graph walks, we find seeds as follows: at the starting point, all vertices are unmarked. We randomly visit a vertex v_i, and mark it. Among its unmarked neighbors, we randomly select a vertex whose degree is lower than v_i. Let v_j be the selected neighbor. Then we mark v_j. We consider v_j's unmarked neighbors, and randomly select one neighbor whose degree is lower than v_j. In this way, we follow down-paths. After we follow a certain number of down-paths, we generate a seed by selecting the final vertex in the path. Then, we mark the seed, and its neighbors. We repeat this procedure until we get k seeds in the graph.

2) Online Weighted Kernel k-means Algorithm: It is known that the standard batch k-means algorithm may yield empty clusters. One way to resolve this problem is by considering the actual objective change when moving data points between clusters [8], [9]. Similar to this approach, GEM computes the actual objective change in weighted kernel k-means, and assigns a vertex to the cluster which yields the greatest decrease in the objective. This procedure is different from the standard batch weighted kernel k-means, and we call this algorithm *Online Weighted Kernel k-means*.

Let us consider a data point $\phi(\hat{\mathbf{x}})$ which is currently in cluster π_p. Suppose that $\phi(\hat{\mathbf{x}})$ is tentatively moved to π_q. Then, the centroid of cluster π_q, denoted by $\mathbf{m}_q$, changes to $\mathbf{m}_q{}^*$ as follows:

$$\mathbf{m}_q{}^* = \mathbf{m}_q + \frac{\hat{w} \cdot \phi(\hat{\mathbf{x}}) - \hat{w} \cdot \mathbf{m}_q}{\sum_{\mathbf{x}_i \in \pi_q} w_i + \hat{w}}. \tag{9}$$

where $\hat{w}$ is the weight of $\phi(\hat{\mathbf{x}})$.

Once $\phi(\hat{\mathbf{x}})$ is moved to π_q, the effective spread of cluster π_q, denoted by J_q, increases to $J_q{}^*$ as follows:

$$J_q{}^* = \sum_{\mathbf{x}_i \in \pi_q} w_i ||\phi(\mathbf{x}_i) - \mathbf{m}_q{}^*||^2 + \hat{w}||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q{}^*||^2. \tag{10}$$

By replacing $\mathbf{m}_q{}^*$ with (9), we can rewrite $J_q{}^*$ as follows:

$$J_q{}^* = J_q + \frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_q} w_i}{\sum_{\mathbf{x}_i \in \pi_q} w_i + \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q||^2. \tag{11}$$

Similarly, when $\phi(\hat{\mathbf{x}})$ moves from π_p to π_q, J_p decreases as follows:

$$J_p{}^* = J_p - \frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_p} w_i}{\sum_{\mathbf{x}_i \in \pi_p} w_i - \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_p||^2. \tag{12}$$

Finally, the movement of $\phi(\hat{\mathbf{x}})$ from π_p to π_q is advantageous if the decrease of J_p is greater than the increase of J_q. That is,

$$\frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_p} w_i}{\sum_{\mathbf{x}_i \in \pi_p} w_i - \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_p||^2 > \frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_q} w_i}{\sum_{\mathbf{x}_i \in \pi_q} w_i + \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q||^2. \tag{13}$$

By considering this condition, GEM determines whether the movement of $\phi(\hat{\mathbf{x}})$ from the current cluster to other cluster will be beneficial or not. If the movement is determined to be profitable, $\phi(\hat{\mathbf{x}})$ is transferred to the cluster for which such a movement yields the greatest decrease of the total objective change. Notice that the greatest decrease of the objective is achieved when we move the vertex to the cluster for which $\frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_q} w_i}{\sum_{\mathbf{x}_i \in \pi_q} w_i + \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q||^2$ is minimum.

Similarly, when we initialize clusters after finding seeds, we assign a vertex to the cluster for which such an assignment allows the least increase of the total objective. Notice that when we assign a new vertex to a cluster, the objective of weighted kernel k-means always increases. If $\phi(\hat{\mathbf{x}})$ is assigned to π_q, the increase of the objective will be as follows:

$$\Delta J_q = \frac{\hat{w} \sum_{\mathbf{x}_i \in \pi_q} w_i}{\sum_{\mathbf{x}_i \in \pi_q} w_i + \hat{w}} ||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q||^2. \tag{14}$$

$\phi(\hat{\mathbf{x}})$ is assigned to π_q such that $\Delta J_q < \Delta J_j$ for all $j \neq q$ $(j = 1, ..., k)$. Recall that we set the weight of each vertex as the degree of the vertex. Also recall that we can compute $||\phi(\hat{\mathbf{x}}) - \mathbf{m}_q||^2$ using (6). Therefore, we can represent (13) and (14) using only graph-theoretic terms. Putting all these together, Figure 3 summarizes the procedure of how GEM initializes clusters after finding seeds, and Figure 4 summarizes the procedure of how GEM applies online weighted kernel k-means algorithm to graph clustering.

Input: $\mathcal{V}$: the vertex set, s_i $(i = 1, ..., k)$: seeds.
Output: $\mathcal{V}_i$ $(i = 1, ..., k)$: initial clusters.

for each cluster $\mathcal{V}_i$ **do**
  $\mathcal{V}_i = s_i$.
end for
for each vertex $\hat{v} \in \mathcal{V}$ **do**
  for each cluster $\mathcal{V}_i$ **do**
    $\hat{\alpha} = \frac{\sigma}{\hat{d}} - \frac{2links(\hat{v}, \mathcal{V}_i)}{\hat{d} \cdot degree(\mathcal{V}_i)} + \frac{links(\mathcal{V}_i, \mathcal{V}_i)}{degree(\mathcal{V}_i)^2}$.
    $\delta_i = \frac{\hat{d} \cdot degree(\mathcal{V}_i)}{degree(\mathcal{V}_i) + \hat{d}} \left\{ \hat{\alpha} + \frac{\sigma}{degree(\mathcal{V}_i)} \right\}$.
  end for
  Find $\mathcal{V}_p$ s.t. $\delta_p \leq \delta_j$ for all j $(j = 1, ..., k)$.
  $\mathcal{V}_p = \{\hat{v}\} \cup \mathcal{V}_p$.
end for

Figure 3. Initialization of clusters

C. Propagation and Refinement

The last step of GEM is to propagate the clustering of the extracted graph to the entire graph, and refine the clustering. Once we get the clustering of the extracted graph, vertices in the extracted graph are considered to be "seeds" for clustering of the entire graph. Starting from these seeds,

Input: $\mathcal{V}$: the vertex set, $\mathcal{V}_i$ $(i = 1, ..., k)$: initial clusters, τ_{max}: maximum number of iterations.
Output: $\mathcal{V}_i^*$ $(i = 1, ..., k)$: final clusters.

Initialize $\tau = 0$.
repeat
  for each vertex $\hat{v} \in \mathcal{V}$ **do**
    $\mathcal{V}_p \longleftarrow$ the current cluster of $\hat{v}$.
    for each cluster $\mathcal{V}_i$ **do**
      $\hat{\alpha} = \frac{\sigma}{\hat{d}} - \frac{2links(\hat{v}, \mathcal{V}_i)}{\hat{d} \cdot degree(\mathcal{V}_i)} + \frac{links(\mathcal{V}_i, \mathcal{V}_i)}{degree(\mathcal{V}_i)^2}$.
      if $\mathcal{V}_i = \mathcal{V}_p$ **then**
        $\delta_i = \frac{\hat{d} \cdot degree(\mathcal{V}_i)}{degree(\mathcal{V}_i) - \hat{d}} \left\{ \hat{\alpha} - \frac{\sigma}{degree(\mathcal{V}_i)} \right\}$.
      else
        $\delta_i = \frac{\hat{d} \cdot degree(\mathcal{V}_i)}{degree(\mathcal{V}_i) + \hat{d}} \left\{ \hat{\alpha} + \frac{\sigma}{degree(\mathcal{V}_i)} \right\}$.
      end if
    end for
    Find $\mathcal{V}_q$ s.t. $\delta_q \leq \delta_j$ for all j $(j = 1, ..., k)$.
    if $\mathcal{V}_p \neq \mathcal{V}_q$ **then**
      $\mathcal{V}_p = \mathcal{V}_p \setminus \{\hat{v}\}$, $\mathcal{V}_q = \mathcal{V}_q \cup \{\hat{v}\}$.
      Update $links(\mathcal{V}_p, \mathcal{V}_p), degree(\mathcal{V}_p)$, $links(\mathcal{V}_q, \mathcal{V}_q), degree(\mathcal{V}_q)$.
    end if
  end for
  $\tau = \tau + 1$.
until not converged and $\tau < \tau_{max}$
$\mathcal{V}_i^* = \mathcal{V}_i$ $(i = 1, ..., k)$.

Figure 4. Graph clustering using online weighted kernel k-means.

we visit the remaining vertices in the original graph in a breadth-first order, and assign the vertices to clusters. In this way, we propagate the clustering of the extracted graph to the original graph, and arrive at the initial clustering of the original graph.

Once all the vertices in the original graph are assigned to some cluster, we refine the clustering of the original graph by applying our online weighted kernel k-means algorithm. Since we have good initial clusters which are achieved by the propagation step, the refinement step efficiently improves the clustering result.

V. Parallel Algorithm

An important advantage of GEM is that it is easy to parallelize for large number of processes. We describe our parallel algorithm of GEM, called PGEM.

The original graph is randomly distributed across different processes such that each process owns an equal-sized subset of vertices and their adjacency lists. We use the term *local vertices* to refer to the subset of vertices each process owns. In the extraction phase, each process scans its local vertices, and picks up the vertices whose degrees are greater than a specified threshold. Then, it extracts edges between the selected vertices to form the subgraph of the original graph. The extracted subgraph is also randomly distributed over all the processes such that each process owns an equal size of the vertex set. Our graph distribution strategy works reasonably well, and we observe that the way the graph is distributed does not lead to a bottleneck.

In the seed selection phase, seeds are generated in rounds. In each round, a certain number of seeds will be generated. Generating a seed corresponds to performing a down-path walk along the graph. In each round, a leader process first decides the number of seeds each process will generate, which is proportional to the number of currently unmarked vertices in that process. Each process randomly picks up a local vertex to start a walk. To advance one step in a walk, a process has to look at the neighbors of the current visited vertex in the walk. Since the subgraph is distributed across processes, the neighbor vertices might be located in a different process. So, we use "Ghost Cells" to access the information of such neighbor vertices: for each remote neighbor vertex, a mirror vertex called ghost cell is maintained. A ghost cell is used to buffer the information of its remote counterpart. In particular, a ghost cell keeps information about whether the vertex is marked or not. Once the information of a vertex is modified, all of its ghost cells will be notified to reflect the new update. In addition, a process can change its ghost cells to modify the corresponding remote vertices.

Each process may choose the next vertex belonging to another process for a walk. Suppose that, during a walk, process p_1 finds that the next vertex it will visit belongs to process p_2. Then process p_1 stops this walk and notifies process p_2 to continue it. When process p_2 receives a walk from process p_1, process p_2 may have already marked the next vertex in the walk. This may be because process p_1 did not receive the update of the ghost cell or process p_2 marked the vertex after process p_1 chose this vertex as the next step for this walk. If the vertex has been marked, process p_2 will send this walk back to the process p_1 to notify that it should choose another vertex as the next step for this walk. Otherwise, process p_2 continues the walk. When a walk is finished, the last visited vertex is selected as a seed by the process currently performing this walk. The seed and its neighbors are marked by that process. To reduce the number of communications between processes, the messages sent by one process (including ghost cells synchronization) are postponed until it finishes processing a certain number of walks so that the messages to the same process are coalesed into a single message.

After all the seeds are generated, each process will be responsible for assigning its local non-seed vertices to clusters. This corresponds to the initialization of clusters. The local unassigned vertices are visited in locally breadth-first order starting from the seeds. The assignment of a vertex can be affected by other processes in two ways: 1) the clusters of its neighbor vertices located remotely may be changed simultaneously; 2) the cluster centroids may be changed by remote processes. To deal with the two cases, we use ghost cells for accessing cluster information of the remote vertices which are neighbors of the local vertices for each process; each process maintains a local copy of all the cluster centroids. Recall that since GEM performs an online variant of the weighted kernel k-means, if a process assigns

a vertex to a cluster, its local copy of the cluster centroid is immediately changed. However, if a process broadcasts the change of a cluster centroid whenever it assigns a vertex, the overall procedure will be serialized. In order to avoid this serialization, we relax the synchronization of the cluster centroids. The cluster centroids are synchronized across processes once a certain number of vertices are visited.

Note that, at the beginning, each cluster contains only one vertex, which is the seed of that cluster. Since the number of seeds is small compared to the number of vertices in the subgraph, the frequency of both the cluster centroid synchronization and the ghost cells synchronization in this phase greatly affects the quality of clusters. We found both need to be synchronized frequently during the assignment of first dozens of vertices. In other words, starting from only one vertex in each cluster, absorbing first several vertices into each cluster is crucial, and the updates of cluster centroids and the cluster assignment of vertices should be shared frequently at this stage. But as the size of each cluster increases to a reasonable size, the frequency of the two synchronizations do not significantly affect the quality of clusters. Therefore, we exponentially decrease the frequency of the two synchronizations as more vertices are visited.

Clustering of the subgraph is performed in rounds. In each round, iterations in weighted kernel k-means are performed in parallel. The parallelization in each round is same as the parallelization of the cluster initialization phase except: 1) each process visits its local vertices in random order. 2) since each cluster has enough vertices in this stage, the delay in updating cluster centroids and ghost cells does not degrade the quality of clusters. Therefore, we synchronize the cluster centroids and ghost cells once a round.

The propagation phase adopts the same parallelization strategy as the cluster initialization phase of the extracted graph except the centroids and ghost cells are not synchronized. We observe this will not affect the cluster quality. Besides, the refinement phase of the original graph adopts the same parallelization strategy as clustering of the subgraph.

VI. Experimental Results

In this section, we evaluate the performance of GEM and PGEM. Our implementation of GEM and PGEM is in C++. PGEM is implemented using MPI. We compare GEM with PMetis, KMetis, and Graclus, and PGEM with ParMetis. PMetis and KMetis are provided by one package, called Metis. We use Metis 5.0.2, Graclus 1.2, and ParMetis 4.0.2. In GEM and PGEM, we extract 10% of the original graph in the extraction phase. For all the datasets, we partition the graph into 100 clusters. We use five different real-world online social networks [10], [11], [12], [13], which are presented in Table I. All the networks are undirected graphs. We performed the sequential experiments on a shared memory machine with AMD Opteron 2.6GHz CPU and 256GB memory. The parallel experiments are done in a large-scale parallel platform at the Texas Advanced Computing Center (TACC), Ranger. Ranger has 3,936 machine nodes, and each node is equipped with a 4×4-core AMD Opteron CPU (the frequency for each core is 2.3GHz) and 32GB memory.

Table I
Summary of social networks

Graph	No. of vertices	No. of edges
Flickr	1,994,422	21,445,057
LiveJournal	1,757,326	42,183,338
Myspace	2,086,141	45,459,079
Twitter (10M)	11,316,799	63,555,738
Twitter (50M)	51,161,011	1,613,892,592

A. Evaluation of GEM

We measure the quality of clusters using two different metrics: the percentage of within-cluster edges, and normalized cut which is defined by (2). Note that a higher percentage of the within-cluster edges, and a lower normalized cut value indicate better quality of clusters. Figure 5 shows the quality of clusters of each algorithm. Graclus fails on Twitter 10M and Twitter 50M graphs. For all the datasets, GEM outperforms PMetis and KMetis in terms of both the within-cluster edges and the normalized cut. For Flickr, GEM produces the best quality clusters among all algorithms. For LiveJournal and Myspace, GEM generates clusters of quality comparable to Graclus.

Figure 6 shows running time of each algorithm. GEM is the fastest algorithm across all the datasets. For Flickr, LiveJournal and Myspace datasets which contain around two million vertices, all the programs finish within 10 minutes. For larger datasets, however, running time significantly increases. For example, KMetis takes about one and half hours to cluster the Twitter 10M graph, and 19 hours to cluster the Twitter 50M graph. For both of these large graphs, GEM is much faster than the other algorithms. GEM takes only about 6 minutes to cluster Twitter 10M graph whereas PMetis takes about 30 minutes. For Twitter 50M graph, GEM takes about three hours whereas PMetis takes about 14 hours.

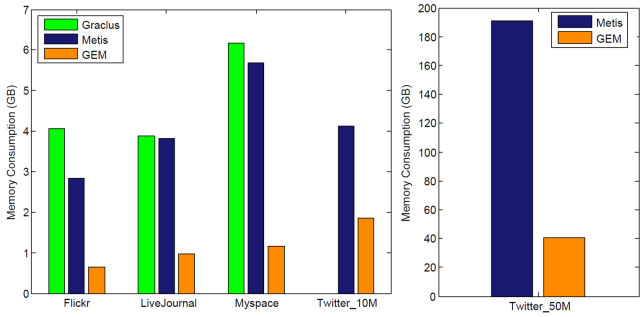
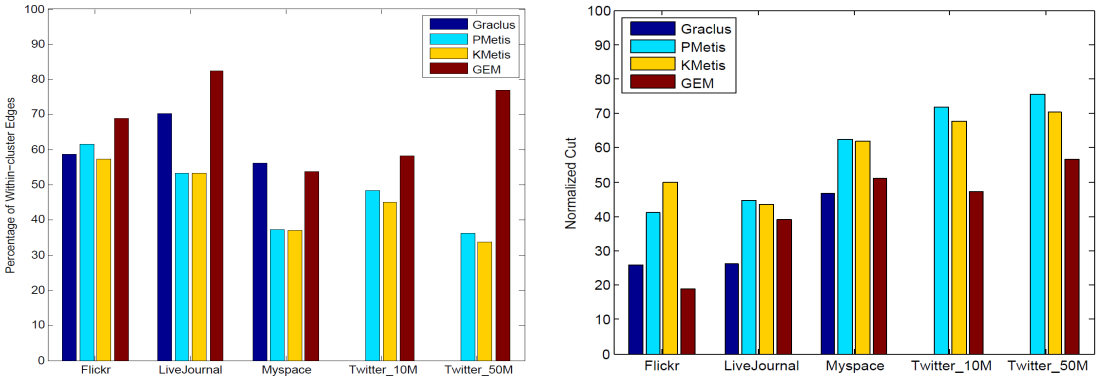


Figure 7. Memory Consumption — GEM consumes much less memory than Graclus and Metis. Since PMetis and KMetis use the same coarsening strategy, we just report the memory consumption for these algorithms as Metis.

Figure 7 shows the memory consumption of each algorithm, i.e., the amount of memory used to store graphs generated during a run of the each algorithm. For multilevel



(a) Percentage of Within-cluster Edges (b) Normalized Cut

Figure 5. Quality of clusters — GEM produces clusters of quality comparable to or better than state-of-the-art algorithms. (a) A higher percentage of within-cluster edges indicates better quality of clusters. (b) A lower normalized cut indicates better quality of clusters.

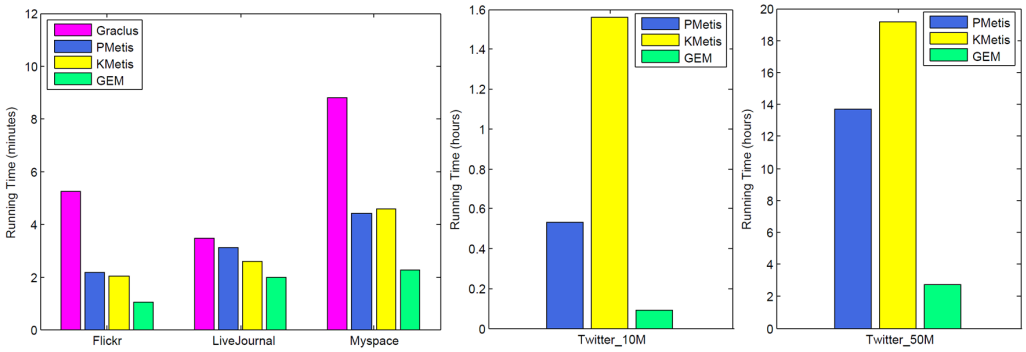


Figure 6. Running Time — GEM is the fastest algorithm across all the datasets. In particular, GEM takes only about 6 minutes to cluster Twitter 10M graph whereas PMetis takes about 30 minutes. GEM takes about three hours to cluster Twitter 50M graph whereas PMetis takes about 14 hours.

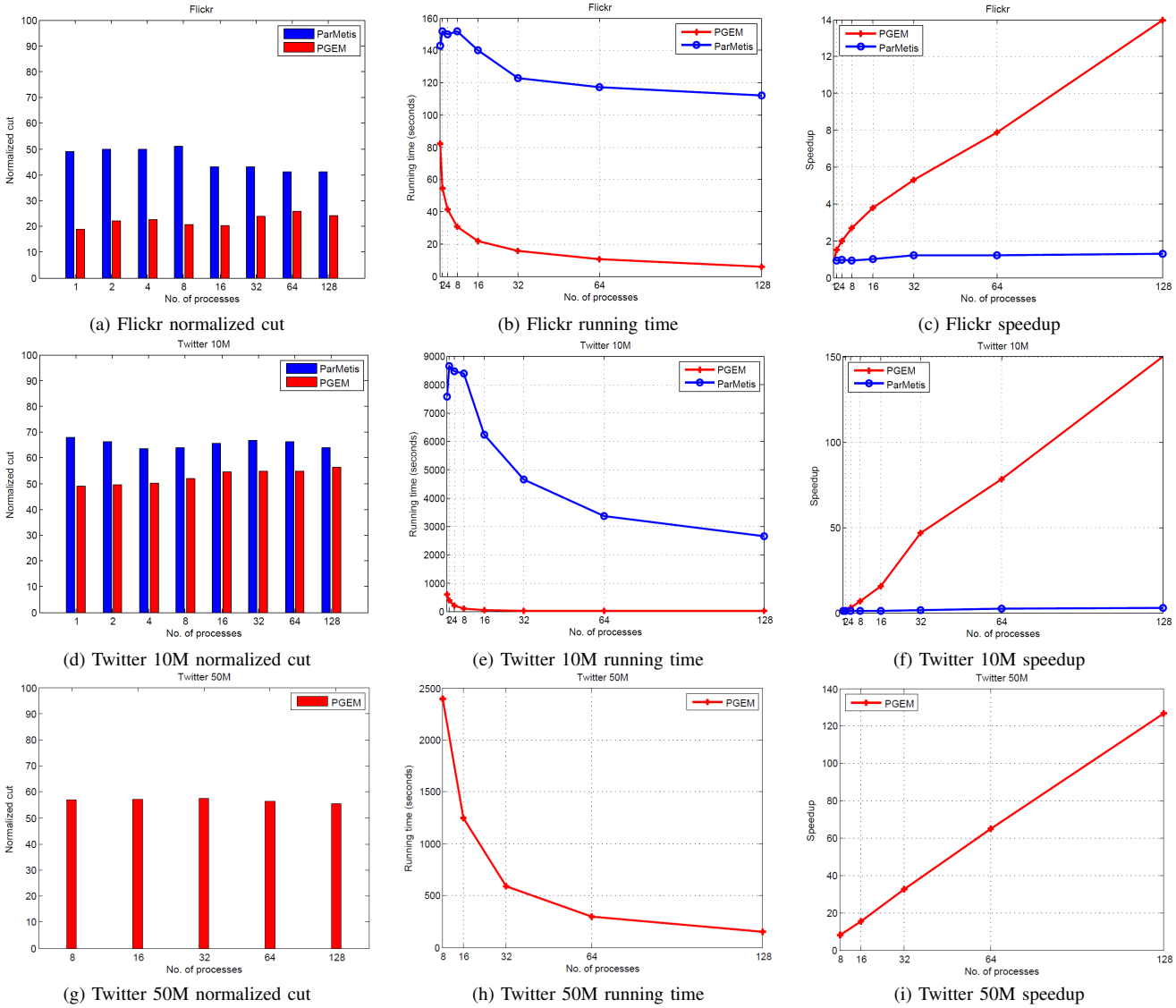
algorithms (Graclus and Metis), we consider memory consumption only up to coarsening phase. Since PMetis and KMetis use the same coarsening strategy, we just report the memory consumption for these algorithms as Metis. Clearly, the memory consumption of GEM is much less than all the other multilevel algorithms. This is because GEM directly extracts a subgraph from the original graph, whereas multilevel algorithms gradually reduce the graph size by multilevel coarsening.

B. Evaluation of PGEM

Figure 8 and Table II show the normalized cut, running time, and speedup of PGEM and ParMetis, according to the number of processes. Speedup is defined as the runtime of the program with one process divided by the runtime with p processes. A higher speedup indicates that the algorithm is more scalable. Due to the memory requirement of Twitter 50M, we use 8 processes as the base for computing the speedup on this graph. PGEM performs consistently better than ParMetis in terms of normalized cut, running time and speedup across all the datasets. In our experiments, ParMetis fails to cluster Twitter 50M graph because the memory is not enough on a single machine node. On Twitter 10M graph, PGEM achieves a speedup of 150 on 128 processes. The super-linear speedup is due to the fact that the run with 128 processes converges faster, i.e., the weighted kernel k-means converges faster. In all cases, the speedup of ParMetis is less than 10, which indicates that the multilevel scheme is hard to be scaled to large number of processes. On Flickr and Myspace, the speedup of PGEM is 14 and 10 on 128 processes, respectively. The lower speedup is because these graphs are small, which means that there is not enough work for each process.

Table II
NORMALIZED CUT (NCUT), RUNTIME (T) AND SPEEDUP ACCORDING TO THE NUMBER OF PROCESSES (P) ON MYSPACE (MS) AND LIVEJOURNAL (LJ)

		PGEM			ParMetis		
	P	NCut	T (sec.)	Speedup	NCut	T (sec.)	Speedup
MS	1	51.0	186.8	1.0	61.4	335.2	1.0
	2	50.0	143.2	1.3	61.6	741.5	0.5
	4	50.7	102.9	1.8	62.2	591.7	0.6
	8	51.3	74.5	2.5	61.7	577.8	0.6
	16	50.9	53.6	3.5	63.1	508.2	0.7
	32	51.0	38.9	4.8	63.5	459.5	0.7
	64	53.4	27.5	6.8	65.0	420.5	0.8
	128	51.9	18.5	10.0	63.7	371.8	0.9
LJ	1	39.1	163.7	1.0	43.8	185.7	1.0
	2	40.0	113.2	1.4	44.7	272,0	0.7
	4	41.5	79.8	2.0	44.7	226.1	0.8
	8	38.5	49.0	3.3	45.0	178.4	1.0
	16	37.4	29.0	5.6	44.9	130.8	1.4
	32	39.9	16.7	9.8	44.2	105.9	1.8
	64	39.7	8.5	19.2	43.8	76.1	2.4
	128	38.8	4.6	35.6	43.3	61.7	3.0



(a) Flickr normalized cut (b) Flickr running time (c) Flickr speedup

(d) Twitter 10M normalized cut (e) Twitter 10M running time (f) Twitter 10M speedup

(g) Twitter 50M normalized cut (h) Twitter 50M running time (i) Twitter 50M speedup

Figure 8. Normalized cut, running time, and speedup on Flickr, Twitter 10M, and Twitter 50M — PGEM produces higher quality clusters and achieves much higher speedups than ParMetis.

Table III
NORMALIZED CUT (NCUT) AND RUNTIME (IN SECONDS) OF PGEM AND PARMETIS ON TWITTER 10M GRAPH WITH DIFFERENT NUMBER OF CLUSTERS

	Best PGEM		Best ParMetis	
No. of Clusters	Ncut	Time (sec.)	NCut	Time (sec.)
64	33.5	3.0	37.8	2,228.5
128	71.5	5.0	82.6	2,280.2
256	142.1	6.7	183.7	2,327.5
512	302.0	11.0	395.4	2,376.4
1024	639.8	14.6	831.1	2,403.0

To see how the clustering results vary with the number of clusters, we run PGEM and ParMetis with different number of clusters on Twitter 10M graph. Table III shows the results. For a given number of clusters, we only present the best result of each algorithm across 1 to 128 processes. In this experiment, PGEM performs significantly better than ParMetis in terms of runtime and normalized cut. All the above experimental results verify that PGEM achieves a significant scalability while generating high quality clusters.

VII. RELATED WORK

To deal with large-scale graph clustering problems, various techniques have been proposed. Satuluri et al. [14] proposed graph sparsification as a way to speed up the existing clustering algorithms. By removing some edges which are likely to be between clusters, they reduced the number of edges in the graph while retaining the community structure of the network. While they derive the clustering result by only considering the sparsified graph which might distort the structure of the original network, GEM propagates the clustering of the representative subgraph to the entire network, and refines the clustering of the original network.

Macropol et al. [15] introduced a technique to find the best clusters in a large network. Instead of seeking exhaustive clusters in a graph, they only found the subset of best clusters to save time and memory by pruning the search space.

Abou-Rjeili et al. [5] proposed new coarsening strategies for multilevel graph clustering framework. They presented new coarsening schemes which allow arbitrary size sets of vertices to be collapsed together. They tried to resolve the graph clustering problem of power law graphs by replacing the traditional coarsening scheme with new coarsening strategies. However, their algorithm still utilizes the multilevel framework, which requires significant time and space complexity than GEM.

Sui et al. [16] presented the first implementation of a clustered low-rank approximation algorithm for large social network graphs, and its application to link prediction. As one part of their parallel clustered low-rank approximation algorithm, they developed a parallel clustering algorithm, called PEK. While PEK is closely related to GEM, PEK uses ParMetis in the initial partitioning phase. Furthermore, PEK has a recursive partitioning phase which guarantees that the largest cluster size is smaller than a certain threshold.

The k-means clustering algorithm has been well studied. Indeed, many different initialization strategies have been proposed. A popular initialization method is MaxMin procedure [17]. The main idea of this procedure is to maximize the distances between seeds. Similar to this method, Arthur et al. [18] proposed the k-means++ algorithm which first randomly selects a seed, and then repeatedly selects seeds with probability proportional to the distance to existing seeds. While both MaxMin procedure and k-means++ require computing all the distances between existing seeds and remaining data points, GEM efficiently finds far away seeds by down-path walk on the graph.

VIII. Conclusion

In this paper, we propose a scalable and memory-efficient graph clustering algorithm, called GEM. Motivated by the observation that the multilevel framework has weaknesses in dealing with large social networks, we develop GEM which allows us to get high quality clusters while substantially saving time and memory. Exploiting two important characteristics of social networks, power law degree distribution, and a hierarchical structure, GEM efficiently extracts a good skeleton graph from the original graph, which is much smaller than the original graph but captures the overall structure of the network well. Once this skeleton graph is clustered using weighted kernel k-means, the clustering results are propagated to the rest of the vertices in the original graph. The main idea of GEM follows the intuition that the clustering of a subgraph consisting of influential people in a social network can be an efficient way to initialize the clustering of the whole network. Experimental results show that GEM produces clusters of quality comparable to or better than state-of-the-art clustering algorithms while it saves much time and memory. The performance gap between GEM and other algorithms is remarkable especially on very large networks with over 10 million vertices. Furthermore, the parallel implementation of GEM, called PGEM, achieves significant scalability while producing high quality clusters.

Acknowledgment

This research was supported by NSF grants CCF-1117055 and CCF-0916309.

References

[1] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, pp. 359–392, 1998.

[2] ——, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, pp. 96–129, 1998.

[3] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[4] G. Karypis and V. Kumar, "A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm," in *SIAM International Conference on Parallel Processing for Scientific Computing*, 1997.

[5] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *IPDPS*, 2006.

[6] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Processing Letters*, vol. 17, 2007.

[7] A. Trusina, S. Maslov, P. Minnhagen, and K. Sneppen, "Hierarchy measures in complex networks," *Physical Review Letters*, vol. 92, 2004.

[8] I. S. Dhillon, Y. Guan, and J. Kogan, "Iterative clustering of high dimensional text data augmented by local search," in *ICDM*, 2002.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2000.

[10] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the Flickr social network," in *The First Workshop on Online Social Networks*, 2008.

[11] H. H. Song, B. Savas, T. W. Cho, V. Dave, Z. Lu, I. S. Dhillon, Y. Zhang, and L. Qiu, "Clustered embedding of massive social networks," in *12th ACM SIGMETRICS*, 2009, pp. 331–342.

[12] "Social Computing Data Repository," http://socialcomputing.asu.edu/datasets/Twitter.

[13] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in twitter: The million follower fallacy," in *ICWSM*, 2010.

[14] V. Satuluri, S. Parthasarathy, and Y. Ruan, "Local graph sparsification for scalable clustering," in *SIGMOD*, 2011.

[15] K. Macropol and A. Singh, "Scalable discovery of best clusters on large graphs," in *VLDB Endowment*, 2010.

[16] X. Sui, T.-H. Lee, J. J. Whang, B. Savas, S. Jain, K. Pingali, and I. Dhillon, "Parallel clustered low-rank approximation of graphs and its application to link prediction," in *LCPC*, 2012.

[17] B. Mirkin, *Clustering for Data Mining: A Data Recovery Approach*. Chapman & Hall/CRC, 2005.

[18] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *ACM-SIAM Symposium on Discrete Algorithms*, 2007.