Fast Algorithm for Modularity-Based Graph Clustering

Hiroaki Shiokawa, Yasuhiro Fujiwara and **Makoto Onizuka**
NTT Software Innovation Center
Nippon Telegraph and Telephone Corporation
3–9–11, Midori–cho Musashino–shi, Tokyo, Japan
{shiokawa.hiroaki, fujiwara.yasuhiro, onizuka.makoto}@lab.ntt.co.jp

Abstract

In AI and Web communities, modularity-based graph clustering algorithms are being applied to various applications. However, existing algorithms are not applied to large graphs because they have to scan all vertices/edges iteratively. The goal of this paper is to efficiently compute clusters with high modularity from extremely large graphs with more than a few billion edges. The heart of our solution is to compute clusters by incrementally pruning unnecessary vertices/edges and optimizing the order of vertex selections. Our experiments show that our proposal outperforms all other modularity-based algorithms in terms of computation time, and it finds clusters with high modularity.

1 Introduction

Graphs can represent data entities as well as the relationships among entities. They arise in a wide range of application domains from the Internet to biological networks and beyond (Fujiwara et al. 2012; Nakatsuji et al. 2012). Graphs are becoming larger and larger, and graphs of unprecedented size can be easily found. For example, the number of monthly active users in Facebook recently exceeded 1 billion (Sibona and Choi 2012). Therefore, there is no longer any doubt about the need for techniques that can analyze large graphs quickly. In these large graphs, graph clustering analysis is one of the more important tools for scientific and industrial data analysis. A graph is divided into groups, called clusters, whose vertices are highly connected inside. By using graph clustering, we can discover the structures and representative examples present in the raw graph data.

Recently, modularity-base clustering proposed by Newman and Girvan (Newman and Girvan 2004) has become one of the most popular algorithms for extracting clusters in a graph. Modularity evaluates the density of edges inside clusters as compared to edges between clusters. The better clustering results are achieved with higher modularity scores. Modularity-based algorithms have been applied to many applications, described below, in AI and Web communities due to its effectiveness:

User recommendation Social tagging systems have emerged as a popular way for users to annotate, and share resources on the Web, such as Yahoo! Delicious and Flickr. However, due to the fast growth of systems, a user is easily overwhelmed by the large amount of data and it is very difficult for the user to dig out the information that he/she is interested in. The modularity-based graph clustering approach suggested by Zhou *et al.* (Zhou et al. 2010), can help users to discover other users with common interests automatically and effectively. It obtains an undirected weighted tag-graph for each user. In each graph, vertices represent the tags used by each user, and edges represent co-occurrences between tag pairs. Then it extracts clusters as the topics which represent a user's interests by using a modularity-based clustering algorithm (Clauset, Newman, and Moore 2004) in each graph. Finally, it computes the interest-based similarity among users by measuring KL–divergence (Kernighan and Lin 1970) of the topics for each user. Their approach can discover users with common interests more effectively than memory-based (Herlocker et al. 1999) and model-based (Hofmann 2004) algorithms.

Event detection Recently, users of microblogging services such as Twitter continuously report their real life events through these services. Detecting life events would be useful for understanding what users are really discussing. Therefore, event detection algorithms have long been a research topic (Yang, Pierce, and Carbonell 1998; Kleinberg 2002). Weng *et al.* applied a modularity-based clustering to wavelet-based signals for event detection (Weng and Lee 2011). They build signals computed by wavelet analysis for individual words, which capture only the bursts in the words' appearance. They then obtain a graph whose vertices and edges represent signals and correlation values between signals, respectively. Finally, they detect events by using the modularity-based algorithm. Their modularity-based approach shows better performance than the LDA-based approach (Blei, Ng, and Jordan 2001) in finding events.

Modularity-based algorithms can also be used in other applications such as image segmentation (Browet, Absil, and Dooren 2011), brain analysis (Yu et al. 2010), and so forth. We omit details due to the space limitations.

Although the modularity-based algorithms are effective for many applications, finding the maximum modularity involves NP-complete complexity (Newman and Girvan 2004). This problem has led to the introduction of approximation approaches. Instead of performing an exhaustive search, a greedy modularity-based approach, named Newman clustering, was proposed by Newman (Newman 2004). It iteratively selects and merges a pair of vertices so as to maximize the rise in modularity. As a result, it produces reasonable clusters with hierarchical structure, which represents the history of merges. Despite the effectiveness in avoiding the NP-complete problem, it requires high computing cost $O(|\mathbb{V}|^2)$, where $|\mathbb{V}|$ is the number of vertices.

Various algorithms have been proposed to reduce the computational cost of Newman clustering (Clauset, Newman, and Moore 2004; Wakita and Tsurumi 2007). Clauset *et al.* proposed a greedy modularity-based algorithm, called CNM (Clauset, Newman, and Moore 2004), which is one of the most widely used methods recently. They used the modularity gain, which is obtained after merging a pair of vertices, and nested heap structures of modularity gain for all pairs of vertices. It iteratively selects and merges the best pair of vertices, which has the largest modularity gain, from the heap until no pairs improve the modularity. The computational cost of CNM is $O(d|\mathbb{E}|\log|\mathbb{V}|)$, where d and $|\mathbb{E}|$ are the depth of the hierarchical clustering result and the number of edges. However, Blondel *et al.* reported that CNM cannot return clustering results in reasonable computational time for graphs with more than 500 thousand vertices (Blondel et al. 2008). Moreover, they also reported that CNM has a tendency to produce super-clusters with significant low modularity, super-clusters contain a large fraction of the vertices, by merging in a global maximization manner.

Blondel *et al.* proposed an efficient greedy algorithm BGLL (Blondel et al. 2008). To the best of our knowledge, BGLL is representative for the state of the art algorithm; it achieves fast clustering with higher modularity than the other algorithms. In contrast to CNM, it computes the modularity gain only for the adjoined vertices pairs as a local maximization. Blondel *et al.* reported that BGLL requires almost 3 hours to process graphs with 118 million vertices (Blondel et al. 2008). Although BGLL is effective for extracting high modularity clusters, it is difficult for BGLL to realize quick responses for graphs of unprecedented size, such as Web graphs with their few billion edges. This is because it iteratively scans all vertices/edges as long as the modularity is incrementing.

To overcome the limitation of computing time in the previous approaches, we propose a novel clustering algorithm. In order to reduce computational cost, we introduce three ideas. First, we incrementally aggregate vertices, which are placed in a same cluster, into a single vertex. Second, we incrementally prune computations of modularity gain for vertices whose clusters are obviously obtained. Last, we optimize the order of vertex selections for efficient clustering. Our proposal has the following attractive characteristics:

- **Efficiency**: The proposed algorithm is considerably faster than existing approaches such as CNM and BGLL.
- **High-modularity**: Our approach provides clustering results with high modularity; it returns almost the same modularity scores as the state of the art approach, BGLL.
- **Effectiveness**: Our algorithm is effective in improving the performance for large-scale complex networks.

To the best of our knowledge, our approach is the first solution to divide graphs into clusters that have more than 100 million vertices and 1 billion edges within 3 minutes. These characteristics confirm the practicality of our algorithm for real world applications. With our proposal, many more applications can be implemented more efficiently.

2 Preliminary

Let $\mathbb{V}$ and $\mathbb{E}$ be sets of vertices and edges, respectively, graph clustering divides graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ into disjoint clusters $\mathbb{C}_i = (\mathbb{V}_i, \mathbb{E}_i)$, in which $\mathbb{V} = \sum_i \mathbb{V}_i$ and $\mathbb{V}_i \cap \mathbb{V}_j = \emptyset$ for any $i \neq j$. To simplify the representations, we assume graphs are undirected and unweighted. However, other types of graphs such as directed and weighted, can be handled with only small modifications.

Modularity, introduced by Newman *et al.* (Newman and Girvan 2004), is widely used to evaluate the cluster structure of a graph from a global perspective. It measures the differences in graph structures from an expected random graph. The main idea of modularity is to find groups of vertices that have a lot of inner-group edges and few inter-group edges. Modularity Q is defined as follows:

Definition 1 (Modularity Q) *Let e_{uv} be the total number of edges between cluster u and v; a_u be the total number of edges that are attached to vertices in cluster u; and m be the total number of edges in the whole graph. The following equation gives the modularity score of the clustering result.*

$$Q = \sum_u \left\{ \frac{e_{uu}}{2m} - \left(\frac{a_u}{2m}\right)^2 \right\}.$$

In Definition 1, $a_u/2m$ is the expected fraction of edges of u, which can be obtained when we assume the graph to be a random graph. Therefore, well clustered graphs will have high modularity scores, since the value of e_{uu} is highly different from the random graph.

3 Proposed method

This section presents details of our proposal. In contrast to all the other algorithms, we can find clusters with high modularity in graphs of unprecedented size, such as more than a few billion of edges, within a few minutes. We give an overview the ideas underlying our algorithm that is followed by a full description including graph clustering algorithm.

3.1 Ideas

We introduce three ideas to avoid the high computation cost of existing algorithms. First, we incrementally aggregate vertices, which are placed in a same cluster, into a single vertex to eliminate unnecessary vertices/edges from the graph. Second, we incrementally prune vertices whose clusters are obtained without modularity computing. Last, we optimize the order of vertex selections to reduce the number of modularity computations in the clustering process. Instead of iterative computations for all vertices/edges in the whole graph, we only compute the key vertices/edges efficiently. Moreover, our proposal successfully produces clustering results with high modularity by obtaining clusters in a local modularity maximization and avoiding skewed access.

These simple ideas have two main advantages. First, we can extract clusters with quite-small computational cost for complex networks (Newman 2003). Our ideas successfully handle the interesting characteristics of complex networks; high clustering coefficients and power-law degree distributions. This is because our ideas are designed to perform well even if the graph has many co-occurrence vertices between adjacent vertices. That is, high clustering coefficients lead our algorithm to compute efficiently. Additionally, our ideas perform well in the case that there are strong imbalances among the degrees of all vertices as in power-law distributions. Thus, our algorithm runs faster on large size complex networks than the state of the art algorithm.

Second, our algorithm can produce clustering results with high modularity by not missing the chances that may improve the modularity. The reason is twofold. First is that our pruning method does not sacrifice modularity. Second is that our ideas successfully prevent our algorithm from producing imbalanced clustering results, which would otherwise greatly degrade the modularity. Therefore, our algorithm can extract clustering results with high modularity.

3.2 Incremental aggregation

We extract clusters by incrementally aggregating vertices placed in a same cluster into an equivalent single vertex with weighted edges. In contrast to previous algorithms, our proposal does not traverse all vertices/edges multiple times. In this section, we formally introduce our incremental aggregation technique and its properties.

We specify the modularity gain proposed by Newman (Newman 2004), since it is also utilized by our algorithm.

Definition 2 (Modularity gain $\triangle Q_{uv}$) *Let $\triangle Q_{uv}$ be the modularity gain which is obtained after merging vertices u and v. The modularity gain $\triangle Q_{uv}$ is defined as follows:*

$$\triangle Q_{uv} = 2\left\{\frac{e_{uv}}{2m} - \left(\frac{a_u}{2m}\right)\left(\frac{a_v}{2m}\right)\right\}.$$

By using modularity gain, our proposal finds clusters in a local maximization manner. When vertex u is selected, it computes the modularity gain of u for each v in $\Gamma(u)$, where $\Gamma(u)$ is the set of vertices neighboring u. After computing all modularity gains between u and v, our algorithm incrementally aggregates u and v that yields the highest rise in modularity. Details of the incremental aggregation and aggregated vertex are as follows:

Definition 3 (Incremental aggregation) *Let vertex v be the neighboring vertex of vertex u that yields the highest rise in modularity. If vertex u has a positive modularity gain for v (i.e. $\triangle Q_{uv} > 0$), the pair of vertices, u and v, are aggregated into a single vertex w. If vertex v has the negative modularity gain (i.e. $\triangle Q_{uv} \leq 0$), the pair of vertices u and v are not aggregated.*

Definition 4 (Aggregated vertex) *We initialize the weight of each edge to 1 in the given graph. If vertex w is aggregated from vertex u and v, vertex w has two types of weighted edges; a self-loop edge and outer edges. The weight of the self-loop edge is obtained by summing (1) weights of the self-loop edges of vertices u and v, and (2) weights of edges between vertex u and v. The weights of outer edges for other vertices are obtained by summing the weights of edges that incident vertices u and v.*

From Definition 4, the degree of w is given as the number of the weighted outer edges that are obtained by aggregating vertices/edges included in the same cluster. Then, we introduce the theoretical properties of Definition 3 and 4.

Lemma 1 (Equivalence of the modularity) *If (1) vertex u and v belong to the same cluster (i.e. $c_u = c_v$) and (2) vertex u and v are aggregated into vertex w, the modularity taken from vertex w is equivalent that of vertices u and v.*

Proof Let $Q_{(u,v)}$ be the modularity for the case that vertex u and v belong to the same cluster, and Q_w be the modularity of aggregated vertex w. From Definition 4, the weighted edges of w are $e_{ww} = e_{uu} + e_{vv} + 2e_{uv} = e_{(u,v)(u,v)}$ and $a_w = a_u + a_v = a_{(u,v)}$. Q_w is obtained as follows:

$$Q_w = \frac{e_{ww}}{2m} - \frac{a_w^2}{4m^2} = \frac{e_{uu}+e_{vv}+2e_{uv}}{2m} - \frac{(a_u+a_v)^2}{4m^2} = Q_{(u,v)}$$

Thus, vertex w has the same modularity as vertices u and v that belong to the same cluster. □

From Lemma 1, we can reduce the number of vertices/edges in the graph without sacrificing modularity quality. Additionally, we advance the following lemma to avoid iterative traversal of all vertices/edges:

Lemma 2 (Negativity of the modularity) *Once a vertex has negative modularity gain for all neighbors, it will never be clustered with its neighbors in the subsequent process.*

Proof We assume vertices v_i and v_j are connected. There are two cases in which the modularity gains of vertex u can be updated. First is that vertex u is connected to both of v_i and v_j. In this case, the modularity gains of u are given as $\triangle Q_{uv_i} < 0$ and $\triangle Q_{uv_j} < 0$, respectively. If v_i and v_j are aggregated into vertex w, the updated modularity gain $\triangle Q_{uw}$ can be obtained by Definition 2 and 4 as follows:

$$\begin{aligned}\triangle Q_{uw} &= 2\left\{\frac{e_{uw}}{2m} - \left(\frac{a_u}{2m}\right)\left(\frac{a_w}{2m}\right)\right\}\\ &= 2\left\{\frac{e_{uv_i}+e_{uv_j}}{2m} - \left(\frac{a_u}{2m}\right)\left(\frac{a_{v_i}+a_{v_j}}{2m}\right)\right\}\\ &= 2\left(\triangle Q_{uv_i} + \triangle Q_{uv_j}\right) < 0.\end{aligned}$$

As can be see, vertex u always has negative modularity gain after merging pairs of neighbor vertices in this case. Next, we assume the case that only vertex v_i is connected to vertex u. In this case, the modularity gain between u and v_i is given as $\triangle Q_{uv_i} < 0$. If v_i and v_j are aggregated into a vertex w, the updated modularity gain $\triangle Q_{uw}$ is obtained as follows:

$$\begin{aligned}\triangle Q_{uw} &= 2\left\{\frac{e_{uw}}{2m} - \left(\frac{a_u}{2m}\right)\left(\frac{a_w}{2m}\right)\right\}\\ &= 2\left\{\frac{e_{uv_i}}{2m} - \left(\frac{a_u}{2m}\right)\left(\frac{a_{v_i}+a_{v_j}}{2m}\right)\right\}\\ &= 2\left(\triangle Q_{uv_i} - \frac{a_u a_{v_j}}{4m^2}\right) < 0.\end{aligned}$$

This case also has no positive improvement of $\triangle Q_{uw}$ after aggregation. Therefore, vertex u never finds a neighbor vertex yielding positive modularity gain. Thus, once a vertex only has negative modularity gains, it will never be clustered in the subsequent process. □

From Lemma 2, we can efficiently reduce the number of traverses for all vertices/edges. This is because, once a vertex is detected as yielding having only negative modularity gain, it is never considered for aggregation thereafter.

Our proposal efficiently handles the structural feature of high clustering coefficient. The clustering coefficient of adjoined vertices (Latapy, Magnien, and Vecchio 2008) measures how close the pair of vertices is to being a clique. By considering this pairwise clustering coefficient, the efficiency of our algorithm is confirmed as follows:

Lemma 3 (Efficiency of incremental aggregation) *Let c be the clustering coefficient of the adjacent vertices, and n be the number of neighbors adjoined to the pair of vertices (i.e. $|\Gamma(u) \cup \Gamma(v)|$). In each incremental aggregation, our algorithm can eliminate cn edges.*

Proof If a pair of vertices have n neighboring vertices, the pair is expected to have cn neighboring vertices that are co-referenced from both of the pair. From Definition 4, cn edges, indicates co-referenced vertices from the pair, will be eliminated by aggregating the pair into a single vertex. Therefore, we can eliminate cn edges in each aggregation.□

From Lemma 3, it is obvious that our algorithm performs well when the given graph has a high clustering coefficient.

3.3 Incremental pruning

In practice, there are a lot of vertices whose clusters are trivially determined, we call them *prunable vertices*. We call the set of vertices whose modularity gains are to be computed as *target vertices*, in the clustering process. Unlike existing algorithms, our algorithm computes the modularity gain for only target vertices by dynamically removing prunable vertices in incremental manner. We formally introduce below the definitions of prunable vertices and target vertices with their theoretical properties. The set of prunable vertices $\mathbb{P}_i$ in the i–th aggregation is defined as follows:

Definition 5 (Prunable vertices) *Let c_u be a cluster to which vertex u belongs. The following equation gives the set of prunable vertices in the i–th aggregation.*

$$\mathbb{P}_i = \begin{cases} \emptyset & (i=0) \\ \{u : |\Gamma(u)| = 1\} \cup \{u : \forall v, w \in \Gamma(u), c_v = c_w\} & (i>0) \end{cases}$$

Definition 5 indicates that a vertex is included in $\mathbb{P}_i$ if (1) the vertex has only a single adjacent vertex, or (2) all the adjacent vertices of the vertex belong to the same cluster. We can introduce the following properties of prunable vertices:

Lemma 4 (Non-negativity of the modularity gain) *If vertex u is included in $\mathbb{P}_i$, the modularity gain of vertex u for each adjacent vertex must be greater than 0.*

Proof From Definition 5, if the vertex included in $\mathbb{P}_i$ has only a single adjacent vertex in the given graph (*i.e.* $|\Gamma(u)| = 1$), we have $e_{uv} = a_u = 1$ and $0 < a_v < 2m$. Therefore, from Definition 2, the modularity gain $\triangle Q_{uv}$ between vertex u and v is always greater than 0. If all neighbor vertices of vertex u belong to the same cluster c_w, (*i.e.* $c_v = c_w, \forall v, w \in \Gamma(u)$), vertex u has edges $e_{uw} = a_u > 0$ and cluster c_w has $0 < a_w < 2m$. Therefore, the modularity gain $\triangle Q_{uw}$ between vertex u and cluster c_w is always greater than 0. □

From Lemma 4, all vertices included in $\mathbb{P}_i$ must have positive modularity gains. That is, all vertices that belong to $\mathbb{P}_i$ are always clustered in their neighbors' cluster. Therefore, we can aggregate these vertices without sacrificing the quality of modularity. From Lemma 4, the set of target vertices in the i–th aggregation, $\mathbb{T}_i$, can be defined as follows:

Definition 6 (Target vertices) *The following equation gives the set of target vertices in the i-th aggregation.*

$$\mathbb{T}_i = \begin{cases} \mathbb{V} & (i=0) \\ \mathbb{T}_{i-1} - \mathbb{P}_i & (i>0) \end{cases}$$

Our algorithm incrementally prunes $\mathbb{P}_i$ from $\mathbb{T}_{i-1}$ in each aggregation step. However, computation costs would be excessive if we naively search for vertices in $\mathbb{P}_i$ such that all of their adjacent vertices belong to a same cluster. For efficient computing, therefore, we introduce a theoretical property of $\mathbb{T}_i$ and $\mathbb{P}_i$ as follows:

Lemma 5 (Incremental pruning) *We can find all vertices included in $\mathbb{P}_i$ by obtaining vertices such that they have only a single adjacent vertex from $\mathbb{T}_{i-1}$ in each aggregation.*

Proof If a vertex in the given graph has only a single adjacent vertex, the vertex can be obviously pruned. If all neighboring vertices of a vertex belong to a same cluster, all of them have already been aggregated into a single vertex by Definition 3. Therefore, we can obtain $\mathbb{P}_i$ by finding vertices such that they have only a single adjacent vertex. □

By Lemma 5, we can efficiently find all vertices in $\mathbb{P}_i$.

3.4 Efficient ordering of vertex selections

We establish efficient ordering of vertex selections for local modularity maximization to reduce the computations. Our proposal dynamically selects vertex with the smallest degree by handling the power-law degree distribution.

One of the famous properties of complex graphs is the power-law degree distribution (Faloutsos, Faloutsos, and Faloutsos 1999); most vertices have relatively few neighbors while a few vertices have many neighbors. Under the power-law degree distribution, the frequency of vertices with degree number of d is proportional to $d^{-\alpha}$, where exponent α is a positive constant that represents the skewness of the degree distribution. A high α implies that the vast majority of vertices have small degree. As α decreases, the graph density and the number of large degree vertices increases. Given the power-law degree distribution, we find the following empirical observation:

- **Observation 1**: Greedy modularity-based algorithms, which maximize modularity in a local manner, can extract clusters with a small number of modularity computations by selecting vertices that have the smallest degree.

We compute the modularity gains of vertex u for all neighbor vertices in $\Gamma(u)$. This process involves $|\Gamma(u)|$ times modularity computations for vertex u to find the vertex that yields the highest rise in modularity gain. Therefore, selecting vertices that have a large degree waste computation

Algorithm 1 Graph clustering

```
Input: $\mathbb{G} = \{\mathbb{V}, \mathbb{E}\}$;
Output: clustering result of $\mathbb{G}$;
 1: $i = 0, \mathbb{P}_0 = \emptyset, \mathbb{T}_0 = \mathbb{V}$;
 2: while $|\mathbb{T}_i| > 0$ do
 3:     $i = i + 1$;
 4:     $\mathbb{P}_i = \{u : |\Gamma(u)| = 1\}$;
 5:     for $\forall u \in \mathbb{P}_i$ do
 6:         aggregate vertex $u$ into its neighbor;
 7:     end for
 8:     $\mathbb{T}_i = \mathbb{T}_{i-1} - \mathbb{P}_i$;
 9:     select vertex $u$ from $\mathbb{T}_i$ that has the smallest degree;
10:     $v = \arg\max_{v'} \triangle Q_{uv'}$;
11:     if $\triangle Q_{uv} > 0$ then
12:         aggregate $u$ and $v$ into a single vertex $w$;
13:         $\mathbb{T}_i = \mathbb{T}_i - \{u, v\}$;
14:         $\mathbb{T}_i = \mathbb{T}_i \cup \{w\}$;
15:     else
16:         $\mathbb{T}_i = \mathbb{T}_i - \{u\}$;
17:     end if
18: end while
```

time. Thus, we find vertices of the highest modularity gain with low computation cost by dynamically selecting vertices with the smallest degree. By combining the ordering and the incremental aggregation, we reduce the size of degrees. Thus, the ordering reduces the computational cost especially for high degree vertex. Additionally, we find the vertex of the highest modularity gain more efficiently as the graph strongly follows power-law degree distribution. This is because vertices in the power-law degree distribution tend to have highly skewed degree.

Moreover, we obtain clusters with high modularity by avoiding skewed access to vertices of large degree. This is because our proposal successfully prevents the results from producing super-clusters, which would otherwise greatly degrade the modularity. Thus, we extract clusters from graphs with high modularity.

3.5 Graph clustering algorithm

Algorithm 1 shows our algorithm. First, if $i = 0$, the algorithm initializes $\mathbb{P}_0 = \emptyset$ and $\mathbb{T}_0 = \mathbb{V}$ based on Definition 5 and 6, respectively (line 1). Next, it incrementally computes prunable vertices $\mathbb{P}_i$ (line 4) and merge each vertex in $\mathbb{P}_i$ into its neighboring cluster (lines 5-7). Next, it obtains target vertices $\mathbb{T}_i$ as described in Lemma 5 and Definition 6 (line 8). It selects vertex u with the smallest degree from $\mathbb{T}_i$ based on Observation 1 (line 9), and finds neighbor vertex v that maximizes the modularity gain as defined in Definition 2 (line 10). If the modularity gain $\triangle Q_{uv}$ is positive, it then aggregates vertices u and v into a single vertex as described in Definition 3 and 4 (lines 11-14). Otherwise, vertex u is pruned from $\mathbb{T}_i$ by Lemma 2 (line 16). If $\mathbb{T}_i$ contains no vertices, it terminates its iteration cycle. Finally, it returns aggregated vertices as a result; all vertices included in an aggregated vertex belong to same cluster.

We provide a theoretical analysis of the computation cost.

Theorem 1 (Computational cost) *Our algorithm requires $O(|\mathbb{E}| - cn|\mathbb{V}|)$ time to obtain a clustering result from a graph, where c and n are the clustering coefficient and the number of neighbors for each adjacent vertices, respectively.*

Proof Our algorithm needs $2|\mathbb{E}|$ computations without the incremental aggregation, because it has to compute $\triangle Q$ for all neighbors for each vertex. Lemma 3 shows that each incremental aggregation eliminates cn edges from the given graph. Moreover, it iterates the incremental aggregations $|\mathbb{T}_i| \approx |\mathbb{V}|$ times by Definition 3 and 4, so we can eliminate $cn|\mathbb{V}|$ edges from the given graph. Therefore, the computation cost can be determined to be $O(|\mathbb{E}| - cn|\mathbb{V}|)$. □

This theorem indicates that the computation cost of our proposal is dramatically smaller than those of existing algorithms; for instance, CNM requires $O(d|\mathbb{E}| \log |\mathbb{V}|)$ to obtain a clustering result. Furthermore, we have even smaller computation cost than the one described in Theorem 1 in practical cases. This is because we have two other techniques to enhance the clustering speed; incremental pruning and efficient ordering. However their computation costs strongly depend on the structure of the graph, we show concrete computation times for real world datasets in the next section.

4 Experimental evaluation

We conducted evaluations to confirm the effectiveness of our algorithm. In the experiments, we used the five public datasets (Boldi et al. 2011) to evaluate our algorithm:

- *dblp-2010*: This scientific collaboration graph was extracted from the bibliography service DBLP in 2010; each vertex is a scientist and each edge is coauthor relationship.
- *ljournal-2008*: This graph was obtained from a social networking site LiveJournal in 2008; each vertex and edge represent a user and friendship among users, respectively.
- *uk-2005*: This graph was obtained from a 2005 crawl of .uk domain; each vertex and edge mean a Web page and a link between pages, respectively.
- *webbase-2001*: This Web graph of .us domain in 2001 was downloaded from the Stanford Webbase project, each vertex and edge mean a Web page and a link, respectively.
- *uk-2007-05*: This graph is a expansion of *uk-2005* crawled from .uk domain Web pages in May, 2007.

The details of our datasets are shown in Table 1, where α is the exponent that controls the skewness of the degree distribution, described in previous section. Additionally, we also use synthetic datasets generated by DIGG[1] to evaluate the effectiveness of our proposal for complex networks. The details setting will be described later.

All experiments were conducted on a Linux 2.6.18 server with Intel Xeon CPU L5640 2.27GHz and 144GB RAM. We implemented our proposal using C++. To evaluate the existing algorithms, we used programs of CNM[2] and BGLL[3] published on their authors' sites.

4.1 Efficiency

We evaluated the clustering performance of each algorithm through wall clock time for each real world dataset. Fig. 1 shows the results of computational time. In Fig. 1, our proposal is tested under two different types; *Proposed* and *Proposed-Limited*. *Proposed* represents the full version of

[1]http://digg.cs.tufts.edu/

[2]http://www.cs.unm.edu/ aaron/research/fastmodularity.htm

[3]https://sites.google.com/site/findcommunities/

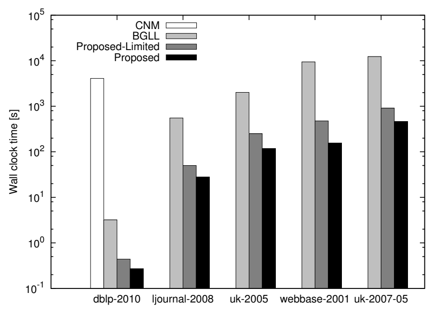

Figure 1: Clustering time

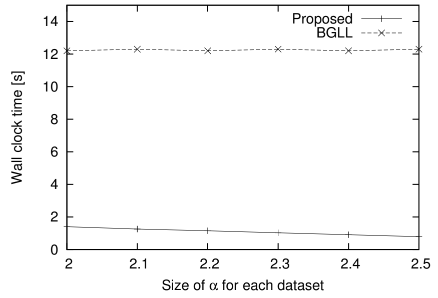

Figure 2: Power-law difference

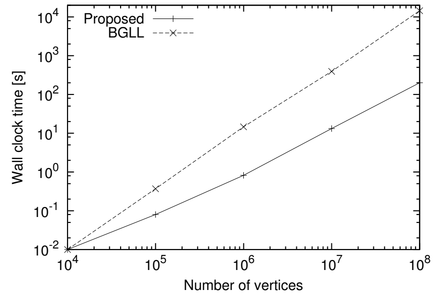

Figure 3: Scalability

Table 1: Datasets

	dblp-2010	ljournal-2008	uk-2005	webbase-2001	uk-2007-05
$\lvert\mathbb{V}\rvert$	326,186	5,363,260	39,459,925	118,142,155	105,896,555
$\lvert\mathbb{E}\rvert$	1,615,400	79,023,142	936,364,282	1,019,903,190	3,738,733,648
α	2.82	2.29	1.71	2.14	1.51

Table 2: Modularity Q

	dblp-2010	ljournal-2008	uk-2005	webbase-2001	uk-2007-05
Proposed	0.90	0.74	0.98	0.98	0.97
BGLL	0.88	0.74	0.97	0.96	0.97
CNM	0.82	-	-	-	-

our algorithm described in Algorithm 1. And *Proposed-Limited* represents the limited version of Algorithm 1 which only uses the incremental aggregation. Since CNM cannot compute clusters in a day except for *dblp*, we omitted the results of CNM for other dataset. Fig. 1 indicates that *Proposed* is significantly faster than the other algorithms under all conditions examined. As described earlier, the existing algorithms traverse all vertices/edges multiple times while our algorithm dynamically eliminates vertices/edges. As a result, our proposal is up to 60 times faster than the state of the art algorithm BGLL; our algorithm computed clusters from the graph with 1 billion edges in 156 seconds. Furthermore, *Proposed-Limited* is up to 20 times faster than BGLL, even though *Proposed* is almost three times faster than *Proposed-Limited* under all conditions. This indicates that the incremental aggregation contributes most to the improvement. *Proposed* more efficiently reduces the computational cost than *Proposed-Limited* by combining incremental aggregation, incremental pruning and efficient ordering.

4.2 Modularity

One major advantage of our algorithm is that it outputs clusters with high modularity. Table 2 shows modularity Q for each of the real world datasets. Table 2 indicates that the modularity score of our algorithm is higher than that of CNM. Since CNM optimizes modularity in a global manner, it tends to produce super-clusters which significantly degrades modularity. In contrast, our algorithm successfully avoid to produce super-clusters by using a local modularity maximization and efficient ordering of vertex selections. Furthermore, Table 2 shows that our proposal achieves slightly higher modularity than BGLL even though BGLL also performs higher modularity than CNM. The computation time of BGLL is significantly larger than ours as shown in Fig. 1. That is, these results show the superiority of our approach over the previous approaches.

4.3 Effectiveness

We evaluate the effectiveness of our algorithm for complex networks that have high cluster coefficients and power-law degree distributions. We compared our proposal with BGLL since it performed well in terms of computing time and modularity. To evaluate the effectiveness, we used synthetic graphs produced by the graph generator DIGG.

Fig. 2 shows the computation times of our proposal and BGLL for different α values (from 2.0 to 2.5) of graphs with 1 million vertices; α represents the skewness of the power-law degree distribution. It is known that the clustering coefficient also follows a power-law degree distribution (Newman 2003); graphs with large α tend to have high clustering coefficients. As shown in Fig. 2, BGLL shows almost constant computational time under all conditions examined. In contrast to BGLL, our algorithm increases its clustering speed as α increases. In the most efficient case, *i.e.* $\alpha = 2.5$, our proposal is up to two times faster than the result of $\alpha = 2.0$. This is because our algorithm eliminates a significant number of vertices/edges as shown in Lemma 3 and 5, when the graph has large α. Thus, our algorithm outperforms BGLL at high α values.

Fig. 3 shows the scalability for our proposal and BGLL; we show the wall clock time as a function of the number of vertices. We varied the number of vertices from 10 thousand to 100 million with $\alpha = 2.5$. As shown in Fig. 3, our algorithm scales better than BGLL. This is because we do not traverse all vertices/edges multiple times. Thus, our proposal clearly achieves higher scalability than BGLL.

5 Conclusion

We have introduced an efficient algorithm for finding clusters with high modularity that allows graphs of unprecedented size to be processed in practical time. Our algorithm is based on three ideas. First, it incrementally aggregates vertices, which are placed in a same cluster, into a single vertex. Second, it incrementally prunes computations for vertices whose clusters can be obtained. Last, it dynamically selects the vertex with the smallest degree. Experiments show that our algorithm can achieve efficient clustering with high modularity. Modularity-based algorithms are fundamental to many current and prospective applications in various disciplines. Our proposal will improve the effectiveness of future applications in AI and Web communities.

References

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2001. Latent Dirichlet Allocation. In *Proceedings of Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems (NIPS 2001)]*, 601–608. MIT Press.

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008:P10008.

Boldi, P.; Rosa, M.; Santini, M.; and Vigna, S. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World Wide Web (WWW 2011)*. ACM Press.

Browet, A.; Absil, P.-A.; and Dooren, P. V. 2011. Community Detection for Hierarchical Image Segmentation. In *Proceedings of the 14th International Workshop on Combinatorial Image Analysis (IWCIA 2011)*, Lecture Notes in Computer Science, 358–371. Springer.

Clauset, A.; Newman, M. E. J.; and Moore, C. 2004. Finding Community Structure in Very Large Networks. *Physical Review E - PHYS REV E* 70:066111.

Faloutsos, M.; Faloutsos, P.; and Faloutsos, C. 1999. On Power-law Relationships of the Internet Topology. In *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 1999)*, 251–262. New York, NY, USA: ACM.

Fujiwara, Y.; Nakatsuji, M.; Yamamuro, T.; Shiokawa, H.; and Onizuka, M. 2012. Efficient Personalized PageRank with Accuracy Assurance. In *Proceedings of 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2012)*, 15–23. ACM.

Herlocker, J. L.; Konstan, J. A.; Borchers, A.; and Riedl, J. 1999. An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999)*, 230–237. ACM.

Hofmann, T. 2004. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)* 22(1):89–115.

Kernighan, B. W., and Lin, S. 1970. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical journal* 49(2):291–307.

Kleinberg, J. M. 2002. Bursty and Hierarchical Structure in Streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, 91–101. ACM.

Latapy, M.; Magnien, C.; and Vecchio, N. D. 2008. Basic Notions for the Analysis of Large Two-mode Networks. *Social Networks* 30(1):31–48.

Nakatsuji, M.; Fujiwara, Y.; Uchiyama, T.; and Toda, H. 2012. Collaborative Filtering by Analyzing Dynamic User Interests Modeled by Taxonomy. In *Proceedings of the 11th International Semantic Web Conference (ISWC 2012)*, Lecture Notes in Computer Science, 361–377. Springer.

Newman, M. E. J., and Girvan, M. 2004. Finding and Evaluating Community Structure in Networks. *Physical Review E - PHYS REV E* 69:026113.

Newman, M. E. J. 2003. The Structure and Function of Complex Networks. *SIAM REVIEW* 45(2):167–256.

Newman, M. E. J. 2004. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E - PHYS REV E* 69:066133.

Sibona, C., and Choi, J. H. 2012. Factors Affecting End–User Satisfaction on Facebook. In *Proceedings of the 6th International AAAI Conference on Weblogs and Social Media (ICWSM 2012)*. AAAI Press.

Wakita, K., and Tsurumi, T. 2007. Finding Community Structure in Mega-Scale Social Networks. In *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, 1275–1276. ACM.

Weng, J., and Lee, B.-S. 2011. Event Detection in Twitter. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media (ICWSM 2011)*. AAAI Press.

Yang, Y.; Pierce, T.; and Carbonell, J. G. 1998. A Study of Retrospective and On-Line Event Detection. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998)*, 28–36. ACM.

Yu, D.; Wu, T.; Shan, Y.; Wang, Y.; He, Y.; and Yang, N. 2010. Making Human Connectome Faster: CPU Acceleration of Brain Network Analysis. In *Proceedings of IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, 593–600. IEEE.

Zhou, T. C.; Ma, H.; Kyu, M. R.; and King, I. 2010. UserRec: A User Recommendation Framework in Social Tagging Systems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press.