Scaling Fine-grained Modularity Clustering for Massive Graphs

Hiroaki Shiokawa , **Toshiyuki Amagasa** and **Hiroyuki Kitagawa**
Center for Computational Sciences, University of Tsukuba, Japan
{shiokawa, amagasa, kitagawa}@cs.tsukuba.ac.jp

Abstract

Modularity clustering is an essential tool to understand complicated graphs. However, existing methods are not applicable to massive graphs due to two serious weaknesses. (1) It is difficult to fully reproduce ground-truth clusters due to the resolution limit problem. (2) They are computationally expensive because all nodes and edges must be computed iteratively. This paper proposes *gScarf*, which outputs fine-grained clusters within a short running time. To overcome the aforementioned weaknesses, gScarf dynamically prunes unnecessary nodes and edges, ensuring that it captures fine-grained clusters. Experiments show that gScarf outperforms existing methods in terms of running time while finding clusters with high accuracy.

1 Introduction

Our work is motivated by one question. How can we efficiently extract fine-grained clusters included in a massive graph? Modularity clustering [Newman, 2004] is a fundamental graph analysis tool to understand complicated graphs [Takahashi *et al.*, 2017; Sato *et al.*, 2018]. It detects a set of clusters that maximizes a clustering metric, *modularity* [Newman, 2004]. Since greedily maximizing the modularity achieves better clustering results, modularity clustering is employed in various AI-based applications [Louni and Subbalakshmi, 2018; Shiokawa *et al.*, 2018].

Although modularity clustering is useful in many applications, it has two serious weaknesses. First, it fails to reproduce ground-truth clusters in massive graphs due to the *resolution limit problem* [Fortunato and Barthélemy, 2007]. Fortunato *et al.* theoretically proved that the modularity becomes larger until each cluster contains $\sqrt{m}$ edges, where m is the number of edges included in a given graph. That is, modularity maximization prefers to find coarse-grained clusters regardless of the ground-truth cluster size. Second, modularity clustering requires a large computation time to identify clusters since it exhaustively computes all nodes and edges included in a graph. In the mid-2000s, modularity clustering was applied to social networks with at most thousands of edges [Palla *et al.*, 2005]. By contrast, recent applications must handle massive graphs with millions or even billions of edges [Louni and Subbalakshmi, 2018] because graphs are becoming larger, and large graphs can be easily found. As a result, current modularity clustering methods [Blondel *et al.*, 2008] need to consume several dozens of hours to obtain clusters from massive graphs.

1.1 Existing Approaches and Challenges

Many studies have strived to overcome these weaknesses. One major approach is to avoid the resolution limit effects by modifying the modularity clustering metric. Locality-aware modularity metrics [Muff *et al.*, 2005; Li *et al.*, 2008; Sankar *et al.*, 2015] are the most successful ones. Modularity is not a scale-invariant since it implicitly assumes that each node interacts with all other nodes. However, it is more reasonable to assume that each node interacts just with its neighbors. By modifying the modularity so that it refers only to neighbor clusters, the metrics successfully moderate the resolution limit effects for small-sized graphs. Costa, however, recently pointed out that graph size still affects such metrics [Costa, 2014]. That is, the metrics do not fully reproduce ground-truth clusters if graphs are large.

Instead of locality aware metrics, Duan *et al.* proposed a sophisticated method, *correlation-aware modularity clustering* (CorMod) [Duan *et al.*, 2014]. They found that modularity shares the same idea with a correlation measure *leverage* [Piatetsky-Shapiro, 1991], and they removed biases that cause the resolution limit problem from modularity through a correlation analysis on leverage. Based on the above analysis, Duan *et al.* defined a different type of modularity named the *likelihood-ratio modularity (LRM)*. Unlike other modularity metrics, CorMod can avoid producing coarse-grained clusters by maximizing LRM in massive graphs.

Although CorMod can effectively reproduce the ground-truth clusters, it still suffers from the large computational costs to handle massive graphs. Similar to the original modularity clustering, CorMod iteratively computes all nodes and edges to find clusters that maximize LRM. This incurs $O(nm \log n)$ time, where n and m are the numbers of nodes and edges, respectively. Several efficient methods have been proposed for modularity maximization such as the node aggregation and pruning approaches used in Louvain [Blondel *et al.*, 2008] and IncMod [Shiokawa *et al.*, 2013], but they cannot be directly applied to CorMod since they have no guarantee for the clustering quality in the LRM maximization.

This is because the approaches focus on only the modularity maximization, whereas CorMod employs more complicated function LRM. Thus, it is a challenging task to improve the computational efficiency of CorMod for fine-grained modularity clustering that approximate the ground-truth clusters.

1.2 Our Approaches and Contributions

We focus on the problem of speeding up CorMod to efficiently extract clusters that well approximate the ground-truth sizes from massive graphs. We present a novel correlation-aware algorithm, *gScarf*, which is designed to handle billion-edge graphs without sacrificing the clustering quality of CorMod. The basic idea underlying gScarf is to dynamically remove unnecessary computations for nodes and edges from the clustering procedure. To determine which computations to exclude, gScarf focuses on the deterministic property of LRM as it is uniquely determined using only the structural properties of clusters such as degrees and cluster sizes. That is, LRM does not need to be computed repeatedly for clusters with the same structural properties.

Based on the deterministic property, gScarf employs the following techniques to improve efficiency. (1) gScarf theoretically derives an incremental form of LRM, namely *LRM-gain*, (2) It introduces *LRM-gain caching* in order to skip unnecessary LRM-gain computations based on the deterministic property, and (3) it employs *incremental subgraph folding* to further improve the clustering speed. As a result, our algorithm has the following attractive characteristics:

- **Efficiency:** gScarf achieves faster clustering than state-of-the-art methods proposed in the last few years (Section 4.1). We theoretically proved that gScarf has a smaller time complexity than the methods (Theorem 1).
- **High accuracy:** gScarf does not sacrifice the clustering quality compared to CorMod, even though gScarf drops computations for nodes and edges (Section 4.2). We theoretically confirmed that our approach does not fail to increase LRM during the clustering procedure.
- **Easy to deploy:** Our approach does not require any user-specified parameters (Algorithm 1). Therefore, gScarf provides users with a simple solution for applications using modularity-based algorithms.

To the best of our knowledge, gScarf is the first solution that achieves high efficiency and fine-grained clustering results on billion-edge graphs. gScarf outperforms the state-of-the-art methods by up to three orders of magnitude in terms of clustering time. For instance, gScarf returns clusters within five minutes for a Twitter graph with 1.46 billion edges. Although modularity clustering effectively enhances application quality, it has been difficult to apply to massive graphs. However, gScarf, which is well suited to massive graphs, should improve the quality in a wider range of AI-based applications.

2 Preliminary

Here, we formally define the notations and introduce the background. Let $G = (V, E, W)$ be an undirected graph, where V, E, and W are sets of nodes, edges, and weights of edges, respectively. The weight of edge $(i, j) \in E$ is initially set to $W_{i,j} = 1$. Graph clustering divides G into disjoint clusters $C_i = (V_i, E_i)$, in which $V = \bigcup_i V_i$ and $V_i \cap V_j = \emptyset$ for any $i \neq j$. We assume graphs are undirected only to simplify the representations. Other types of graphs such as directed graphs can be handled with only slight modifications. For further discussions, we detail how to apply our proposed approaches to the directed graphs in Appendix A.

2.1 Modularity

The modularity-based algorithms detect a set of clusters $\mathbb{C}$ from G so that $\mathbb{C}$ maximizes a clustering metric, called *modularity* [Newman, 2004]. Modularity measures the differences in graph structures from an expected random graph. Here, we denote $tp(i) = e_i/2m$ and $ep(i) = (a_i/2m)^2$, where e_i is a number of edges within C_i, a_i is a total degree of all nodes in C_i, and m is the number of edges in G. Given a set of clusters $\mathbb{C}$, modularity for $\mathbb{C}$ is given by the following function $Q(\mathbb{C})$:

$$Q(\mathbb{C}) = \sum_i Q(C_i) = \sum_i \{tp(i) - ep(i)\}\,. \qquad (1)$$

Note that, in Equation (1), $ep(i) = (a_i/2m)^2$ indicates the expected fraction of edges, which is obtained by assuming that G is a random graph. Thus, Q increases when each cluster C_i has a larger fraction of inner-edges $ep(i) = e_i/2m$ than that of the random graph.

Fortunato and Barthelémy theoretically concluded that modularity suffers from the resolution limit problem [Fortunato and Barthélemy, 2007]. They proved that $Q(\mathbb{C})$ increases until each cluster includes $\sqrt{2m}$ edges. That is, if a given graph is too large, the modularity maximization produces many super-clusters. Thus, modularity maximization fails to reproduce the ground-truth clusters in massive graphs.

2.2 Likelihood-ratio Modularity

To overcome the problem, Duan *et al.* recently proposed *CorMod* [Duan *et al.*, 2014]. CorMod employs the following modularity-based metric *likelihood-ratio modularity* (LRM), which guarantees to avoid the resolution limit:

$$LRM(\mathbb{C}) = \sum_i LRM(C_i) = \sum_i \frac{Pr(tp(i), e_i, 2m)}{Pr(ep(i), e_i, 2m)}, \qquad (2)$$

where $Pr(p, k, n)$ is $\binom{n}{k} p^k (1-p)^{n-k}$, which denotes the binomial probability mass function. $Pr(tp(i), e_i, 2m)$ shows the probability of obtaining C_i from G, whereas $Pr(ep(i), e_i, 2m)$ is the expected probability that C_i is taken from a random graph. As shown in Equation (2), LRM is designed to balance the size of $Pr(tp(i), e_i, 2m)$ and the ratio of $\frac{Pr(tp(i), e_i, 2m)}{Pr(ep(i), e_i, 2m)}$ for each cluster. If C_i is small, $Pr(tp(i), e_i, 2m)$ becomes large while the ratio of $\frac{Pr(tp(i), e_i, 2m)}{Pr(ep(i), e_i, 2m)}$ becomes small. On the other hand, if C_i is large, $Pr(tp(i), e_i, 2m)$ becomes small, the ratio becomes large. Thus, unlike modularity, LRM successfully avoids to produce super-clusters regardless of the graph sizes.

3 Proposed Method: gScarf

We present gScarf that efficiently detects clusters from massive graphs based on LRM maximization. First, we overview the ideas underlying gScarf and then give a full description.

3.1 Ideas

Our goal is to efficiently find clusters without sacrificing clustering quality compared with CorMod. CorMod, which is a state-of-the-art approach, iteratively computes LRM for all pairs of clusters until LRM no longer increases. By contrast, gScarf skips unnecessary LRM computations by employing the following approaches. First, we theoretically derive an incremental form of LRM named *LRM-gain*. LRM-gain is a deterministic criterion to measure the rise of LRM obtained after merging a pair of clusters. Second, we introduce *LRM-gain caching* to remove duplicate LRM computations that are repeatedly invoked. Finally, we employ *incremental subgraph folding* to prune unnecessary computations for nodes and edges. Instead of exhaustive computations for the entire graph, gScarf computes only the essential nodes and edges efficiently to find clusters.

Our ideas have two main advantages. (1) gScarf finds all clusters with a quite-small running time on real-world graphs. Our ideas successfully handle the power-law degree distribution, which is a well-known property of real-world graphs [Faloutsos *et al.*, 1999]. This is because gScarf increases its performance if a lot of nodes have similar degrees. Hence, the above property lead gScarf to compute efficiently. (2) gScarf produces almost the same clusters as those of CorMod [Duan *et al.*, 2014]. We theoretically demonstrate that gScarf does not miss chances to improve LRM, although gScarf dynamically prunes nodes and edges from a graph. Thus, gScarf does not sacrifice clustering quality compared to CorMod.

3.2 Incremental LRM Computation

We define *LRM-gain*, which measures the rise of the LRM scores obtained after merging two clusters.

Definition 1 (LRM-gain $\triangle L_{i,j}$). *Let $\triangle L_{i,j}$ be the gain of LRM obtained by merging two clusters, C_i and C_j, $\triangle L_{i,j}$ is given as follows:*

$$\triangle L_{i,j} = \triangle P_{i,j} - \triangle Q_{i,j}, \tag{3}$$

where $\triangle P_{i,j}$ and $\triangle Q_{i,j}$ are the gains of the probability ratio and the modularity, respectively. Let $C_{(i,j)}$ be the cluster obtained by merging C_i and C_j, and their definitions are given as follows:

$$\triangle P_{i,j} = P(C_{(i,j)}) - P(C_i) - P(C_j), \tag{4}$$

$$\triangle Q_{i,j} = Q(C_{(i,j)}) - Q(C_i) - Q(C_j), \tag{5}$$

where $P(C_i)$ is $tp(i) \ln \frac{tp(i)}{ep(i)}$ if $tp(i) > 0$; otherwise, 0.

Note that $\triangle P_{i,j}$ represents a gain of the probability-ratio [Brin *et al.*, 1997], which is a well-known correlation measure. Additionally, $\triangle Q_{i,j}$ is the modularity-gain [Clauset *et al.*, 2004] that measures the increase of the modularity after merging C_i and C_j into the same cluster $C_{(i,j)}$. From Definition 1, we have the following property:

Lemma 1. *Let $C_{(i,k)}$ and $C_{(j,k)}$ be clusters obtained by merging C_k into C_i and C_j, respectively. We always have $LRM(C_{(i,k)}) \geq LRM(C_{(j,k)})$ iff $\triangle L_{i,k} \geq \triangle L_{j,k}$.*

Proof. We first transform Equation (2) by applying the Poisson limit theorem [Papoulis and Pillai, 2002] to the binomial probability mass functions, $Pr(tp(i), e_i, 2m)$ and $Pr(ep(i), e_i, 2m)$. Since $tp(i)$ and $ep(i)$ are significantly small, we can transform $LRM(C_i)$ as follows:

$$LRM(C_i) = \frac{Pr(tp(i),e_i,2m)}{Pr(ep(i),e_i,2m)} = \frac{(2m\cdot tp(i))^{2m\cdot tp(i)}\cdot \mathrm{e}^{-2m\cdot tp(i)}}{(2m\cdot ep(i))^{2m\cdot tp(i)}\cdot \mathrm{e}^{-2m\cdot ep(i)}}$$
$$= \left(\frac{tp(i)}{ep(i)}\right)^{2m\cdot tp(i)} \cdot \mathrm{e}^{-2m\cdot Q(C_i)}. \tag{6}$$

By letting $L(C_i) = \frac{1}{2m} \ln LRM(C_i)$, we have,

$$L(C_i) = tp(i) \ln \frac{tp(i)}{ep(i)} - \{tp(i) - ep(i)\}. \tag{7}$$

Clearly, $\triangle L_{i,k} = L(C_{(i,k)}) - L(C_i) - L(C_k)$.

We then prove $LRM(C_{(i,k)}) \geq LRM(C_{(j,k)}) \Rightarrow \triangle L_{i,k} \geq \triangle L_{j,k}$. Since $LRM(C_{(i,k)}) \geq LRM(C_{(j,k)})$, we clearly have $LRM(C_{(i,k)}) - LRM(C_i) - LRM(C_k) \geq LRM(C_{(j,k)}) - LRM(C_j) - LRM(C_k)$. Thus, we hold $\frac{LRM(C_{(i,k)})}{LRM(C_i)LRM(C_k)} \geq \frac{LRM(C_{(j,k)})}{LRM(C_j)LRM(C_k)}$. By applying $L(C_i) = \frac{1}{2m} \ln LRM(C_i)$ for the above inequality, $L(C_{(i,k)}) - L(C_i) - L(C_k) \geq L(C_{(j,k)}) - L(C_j) - L(C_k)$. Hence, $LRM(C_{(i,k)}) \geq LRM(C_{(j,k)}) \Rightarrow \triangle L_{i,k} \geq \triangle L_{j,k}$.

We omit the proof of $\triangle L_{i,k} \geq \triangle L_{j,k} \Rightarrow LRM(C_{(i,k)}) \geq LRM(C_{(j,k)})$. However, it can be proved in a similar fashion of the above proof. □

Lemma 1 implies that the maximization of LRM and LRM-gain are equivalent. Consequently,gScarf can find clusters that increase LRM by using LRM-gain in a local maximization manner. We also identify two additional properties that play essential roles to discuss our LRM-gain caching technique shown in Section 3.3.

Lemma 2. *$\triangle L_{i,j}$ is uniquely determined by e_i, a_i, e_j, a_j, and $e_{i,j}$, where $e_{i,j}$ is the number of edges between i and j.*

Proof. Since $tp((i,j)) = \frac{e_i+2e_{i,j}+e_j}{2m}$ and $ep((i,j)) = \left(\frac{a_i+a_j}{2m}\right)^2$, we can obtain $\triangle P_{i,j}$ from e_i, a_i, e_j, a_j, and $e_{i,j}$. Furthermore, since $\triangle Q_{i,j} = 2\{\frac{e_{i,j}}{2m} - (\frac{a_i}{2m})(\frac{a_j}{2m})\}$ [Blondel *et al.*, 2008], $\triangle Q_{i,j}$ is clearly determined by a_i, a_j, and $e_{i,j}$. Hence, we hold Lemma 2 by Definition 1. □

Lemma 3. *For each pair of clusters C_i and C_j, it requires $O(1)$ time to compute its LRM-gain $\triangle L_{i,j}$.*

Proof. Lemma 3 is trivial from Definition 1. □

3.3 LRM-gain Caching

We introduce *LRM-gain caching* to reduce the number of computed nodes and edges. As shown in Lemma 2, LRM-gain $\triangle L_{i,j}$ is uniquely determined by a tuple of structural properties $s_{i,j} = \langle e_i, a_i, e_j, a_j, e_{i,j}\rangle$. That is, if $s_{i,j}$ equals to $s_{i',j'}$, then $\triangle L_{i,j} = \triangle L_{i',j'}$. Hence, once $\triangle L_{i,j}$ and its corresponding structural property $s_{i,j}$ obtained, $\triangle L_{i,j}$ can be reused to compute other pairs of clusters whose structural properties are equivalent to $s_{i,j}$.

To leverage the above deterministic property, gScarf employs *LRM-gain caching* given as follows:

Definition 2 (LRM-gain caching)**.** *Let h be a hash function that stores $\triangle L_{u,v}$ with its corresponding structural property $s_{u,v}$. The LRM-gain caching $h(s_{i,j})$ for a structural property $s_{i,j} = \langle e_i, a_i, e_j, a_j, e_{i,j} \rangle$ is defined as follows:*

$$h(s_{i,j}) = \begin{cases} \triangle L_{i',j'} & (\textit{If gScarf finds } s_{i',j'} \equiv s_{i,j} \textit{ in } h), \\ \textit{null} & (Otherwise), \end{cases} \quad (8)$$

where $s_{i',j'} \equiv s_{i,j}$ denotes $s_{i',j'}$ is equivalent to $s_{i,j}$.

gScarf skips the computation of $\triangle L_{i,j}$ if $s_{i',j'} \equiv s_{i,j}$ is in h; otherwise, gScarf computes $\triangle L_{i,j}$ by Definition 1.

To discuss the theoretical aspect of Definition 2, we introduce a well-known property called the *power-law degree distribution* [Faloutsos *et al.*, 1999]. Under this property, the frequency of nodes with k neighbors is $p(k) \propto k^{-\gamma}$, where exponent γ is a positive constant that represents the skewness of the degree distribution.

Given the power-law degree distribution, LRM-gain caching has the following property according to Lemma 3:

Lemma 4. *LRM-gain caching requires $O(d^{2\gamma})$ time during the clustering procedure, where d is the average degree.*

Proof. To simplify, we assume that each cluster is a singleton, *i.e.* $C_i = \{i\}$ for any $i \in V$. That is, $s_{i,j} \equiv s_{i',j'}$ iff $a_i = a_{i'}$ and $a_j = a_{j'}$ because we always have $e_i = e_j = 0$ and $e_{i,j} = 1$ for the singleton clusters. Since the frequency of nodes with k neighbors is $p(k) \propto k^{-\gamma}$, the expected number of pairs whose structural properties are equivalent to $s_{i,j}$ is $2m \cdot p(a_i)p(a_j)$. Thus, to cover the whole of a graph, gScarf needs to compute LRM-gain scores at least $O(\frac{2m}{2m \cdot p(a_i)p(a_j)}) \approx O(\frac{1}{p(d)^2})$ times during the clustering. From Lemma 3, each LRM-gain computation needs $O(1)$ time, LRM-gain caching, therefore, requires $O(\frac{1}{p(d)^2}) = O(d^{2\gamma})$ times. □

3.4 Incremental Subgraph Folding

We introduce *incremental subgraph folding* that removes unnecessary computations. Real-world graphs generally have high clustering coefficient [Watts and Strogatz, 1998]; they have a lot of triangles in a graph. However, a high clustering coefficient entails unnecessary computations in CorMod since the triangles create numerous duplicate edges between clusters. To avoid such computations, gScarf incrementally folds nodes placed in the same cluster into an equivalent node with weighted edges by extending theoretical aspects of the incremental aggregation presented in [Shiokawa *et al.*, 2013].

First, we define the subgraph folding as follows:

Definition 3 (Subgraph Folding)**.** *Given $G = (V, E, W)$, $i \in C_i$ and $j \in C_j$, where $C_i = (V_i, E_i)$ and $C_j = (V_j, E_j)$. Let f be a function that maps every nodes in $V \backslash \{V_i \cup V_j\}$ to itself; otherwise, maps it to a new node x. The subgraph folding of nodes i and j results in a new graph $G' = (V', E', W')$, where $V' = V \backslash \{V_i \cup V_j\} \cup \{x\}$ and $E' = \{(f(u), f(v)) | (u, v) \in E\}$ with updated weight values $W'_{f(u),f(v)}$ such that*

$$W'_{f(u),f(v)} = \begin{cases} 2W_{u,v} + W_{u,u} + W_{v,v} & (f(u) = x, f(v) = x) \\ W_{i,v} + W_{j,v} & (f(u) = x, f(v) \neq x) \\ W_{u,i} + W_{u,j} & (f(u) \neq x, f(v) = x) \\ W_{u,v} & (f(u) \neq x, f(v) \neq x) \end{cases}.$$

Definition 3 transforms two nodes i and j into an equivalent single node x with weighted edges. It replaces multiple edges between i and j into a self-loop edge $(x, x) \in E'$ with a weight value $W_{x,x}$, which represents the number of edges between i and j. Similarly, let $k \in \Gamma(x)$, it merges edges (i, k) and (j, k) into weighted single edge (x, k). Thus, the subgraph folding reduces the number of nodes/edges from G.

Lemma 5. *Let nodes $i, j \in V$ be in the same cluster (i.e. $C_i = C_j$). If gScarf folds the nodes i and j into a new node w by Definition 3, then LRM taken from node w is equivalent to that of the $C_{(i,j)}$ composed of nodes i and j.*

Proof. Recall Equation (7), we have $L(C_w) = tp(w) \ln \frac{tp(w)}{ep(w)} - \{tp(w) - ep(w)\}$. Since gScarf folds the nodes i and j into node w, $e_w = e_i + e_j + 2e_{i,j}$ and $a_w = a_i + a_j$ by Definition 3.Thus, we have

$$\begin{aligned} L(C_w) &= tp(w) \ln \tfrac{tp(w)}{ep(w)} - \{tp(w) - ep(w)\} \\ &= \tfrac{e_i + e_j + 2e_{i,j}}{2m} \ln \frac{\frac{e_i + e_j + 2e_{i,j}}{2m}}{\left(\frac{a_i + a_j}{2m}\right)^2} - \tfrac{e_i + e_j + 2e_{i,j}}{2m} + \left(\frac{a_i + a_j}{2m}\right)^2 \\ &= \tfrac{e_{(i,j)}}{2m} \ln \frac{\frac{e_{i,j}}{2m}}{\left(\frac{a_{(i,j)}}{2m}\right)^2} - \tfrac{e_{(i,j)}}{2m} + \left(\tfrac{a_{(i,j)}}{2m}\right)^2 = L(C_{(i,j)}). \quad (9) \end{aligned}$$

By Lemma 1, if $L(C_w) = L(C_{(i,j)})$, then we have $LRM(C_w) = LRM(C_{(i,j)})$. Therefore, from Equation (9), we hold Lemma 5. □

From Lemma 5, the subgraph folding does not fail to capture LRM-gain that shows a positive score. That is, gScarf reduces the number of duplicated edges without sacrificing the clustering quality of CorMod.

Based on Definition 3, gScarf performs subgraph folding in an incremental manner during the clustering procedure. gScarf greedily searches clusters using LRM-gain shown in Definition 1. Once cluster C_i is chosen, gScarf computes LRM-gain $\triangle L_{i,j}$ for all neighbor clusters $C_j \in \Gamma(C_i)$, where $\Gamma(C_i)$ denotes a set of clusters neighboring C_i. After that, gScarf folds a pair of clusters that yields the largest positive LRM-gain based on Definition 3.

Our incremental subgraph folding has the following theoretical property in terms of the computational costs.

Lemma 6. *A subgraph folding entails $O(d)$ times, where d is the average degree.*

Proof. By Definition 3, the subgraph folding requires to update the weights of each edge by traversing all neighboring clusters. Thus, a subgraph folding for C_i and C_j entails time-complexity $O(\min\{|\Gamma(C_i)|, |\Gamma(C_j)|\}) = O(d)$. □

3.5 Algorithm

Algorithm 1 gives a full description of gScarf. First, gScarf initializes each node as a singleton cluster, *i.e.* $C_i = \{i\}$, and it stores all clusters into a target node set $\mathbb{T}$ (lines 1-3). Then, gScarf starts the clustering phase to find a set of clusters $\mathbb{C}$ that maximizes LRM in a local maximization manner (lines 4-15). Once gScarf selects a cluster C_i from $\mathbb{T}$ (line 5), it explores neighboring cluster C_{best} that yields the largest

Algorithm 1 Proposed method: gScarf

```
Input: A graph $G = (V, E, W)$;
Output: A set of clusters $\mathbb{C}$;
 1: $\mathbb{T} = \emptyset$;
 2: for each $i \in V$ do
 3:     $C_i = \{i\}$, and $\mathbb{T} = \mathbb{T} \cup \{C_i\}$;
 4: while $\mathbb{T} \neq \emptyset$ do
 5:     Get $C_i$ from $\mathbb{T}$;
 6:     $C_{best} = C_i$;
 7:     for $C_j \in \Gamma(C_i)$ do
 8:         $s_{i,j} = \langle e_i, a_i, e_j, a_j, e_{i,j} \rangle$;
 9:         if $h(s_{i,j}) = null$ then $h(s_{i,j}) \leftarrow \triangle L_{i,j}$;
10:         if $h(s_{i,j}) > h(s_{i,best})$ then $C_{best} = C_j$;
11:     if $h(s_{i,best}) > 0$ then
12:         $C' \leftarrow fold(C_i, C_{best})$ by Definition 3;
13:         $\mathbb{T} = \mathbb{T} \backslash \{C_i, C_{best}\} \cup \{C'\}$;
14:     else $\mathbb{T} = \mathbb{T} \backslash \{C_i\}$;
15: return $\mathbb{C}$;
```

positive score of LRM-gain (lines 6-10). To improve the clustering efficiency, gScarf uses the LRM-gain caching $h(s_{i,j})$ in Definition 2; gScarf skips unnecessary computations based on the deterministic property shown in Lemma 2 (lines 8-10). If gScarf finds the structural property $s_{i',j'} \equiv s_{i,j}$ in h, then it reuses $h(s_{i',j'}) = \triangle L_{i',j'}$ instead of $\triangle L_{i,j}$; otherwise, it computes $\triangle L_{i,j}$ by Definition 1 (line 9). After finding C_{best}, gScarf performs the incremental subgraph folding for C_i and C_{best} (lines 11-14). If $h(s_{i,best}) = \triangle L_{i,best} > 0$, gScarf contracts C_i and C_{best} into a single cluster C' by Definition 3 (line 19). gScarf terminates when $\mathbb{T} = \emptyset$ (line 4). Finally, it outputs a set of clusters $\mathbb{C}$ in (line 15).

The computational cost of gScarf is analyzed as follows:

Theorem 1. *gScarf incurs $O(m + d^{2\gamma})$ time to obtain a clustering result from a graph G, where m is the number of edges in G, d is the average degree, and γ is a positive constant that controls the skewness of the degree distribution.*

Proof. Algorithm 1 requires $O(|\mathbb{T}|) \approx O(n)$ iterations for the clustering phase. In each iteration, gScarf invokes at most one subgraph folding (lines 18-21) that entails $O(d)$ costs from Lemma 6. Thus, it has $O(nd) = O(m)$ times to perform the clustering phase. Additionally, gScarf employs the LRM-gain caching during the iterations. As we proved in Lemma 4, it incurs $O(d^{2\gamma})$ times. Therefore, gScarf requires $O(m + d^{2\gamma})$ times to obtain a clustering result. □

The computational costs of gScarf are dramatically smaller than those of CorMod, which requires $O(nm \log n)$ time. In practice, real-world graphs show $d^{2\gamma} \ll m$ since they generally have small values of d and γ due to the power-law degree distribution [Faloutsos *et al.*, 1999]. Specifically, as shown in Table 1, the real-world datasets examined in the next section have $d < 39$ and $\gamma < 2.3$. Thus, gScarf has a nearly linear time complexity against the graph size (*i.e.,* $O(m + d^{2\gamma}) \approx O(m)$) on the real-world graphs with the power-law degree distribution. Furthermore, our incremental subgraph folding effectively handles the clustering coefficient [Watts and Strogatz, 1998; Shiokawa *et al.*, 2015] to further reduce the computational costs. Thus, gScarf has an even smaller cost than the one shown in Theorem 1.

Name	n	m	d	γ	Ground-truth	Source
YT	1.13 M	2.98 M	2.63	1.93	✓	com-Youtube (SNAP)
WK	1.01 M	25.6 M	25.1	2.02	N/A	itwiki-2013 (LAW)
LJ	3.99 M	34.6 M	8.67	2.29	✓	com-LiveJournal (SNAP)
OK	3.07 M	117 M	38.1	1.89	✓	com-Orkut (SNAP)
WB	118 M	1.01 B	8.63	2.14	N/A	webbase-2001 (LAW)
TW	41.6 M	1.46 B	35.2	2.27	N/A	twitter-2010 (LAW)

Table 1: Statistics of real-world datasets.

4 Experimental Evaluation

In this section, we experimentally evaluate the efficiency and the clustering accuracy of gScarf.

We compared gScarf with the following state-of-the-art graph clustering baseline algorithms:

- **CorMod**: The original correlation-aware modularity clustering [Duan *et al.*, 2014]. CorMod greedily maximizes the same criteria (LRM) as gScarf.
- **Louvain**: The most popular modularity clustering for large graphs [Blondel *et al.*, 2008]. This method greedily maximizes the modularity in Equation (1).
- **IncMod**: The fastest modularity maximization method proposed by [Shiokawa *et al.*, 2013]. It improves the efficiency of Louvain via incremental aggregation and pruning techniques.
- **pSCAN**: The density-based graph clustering recently proposed by [Chang *et al.*, 2017]. It extracts clusters by measuring a density of node connections based on thresholds ϵ and μ. We set $\epsilon = 0.6$ and $\mu = 5$ as the same settings as used in [Chang *et al.*, 2017].
- **CEIL**: A scalable resolution limit-free method based on a new clustering metric that quantifies internal and external densities of the cluster [Sankar *et al.*, 2015]. It detects clusters by greedily maximizing the metric.
- **MaxPerm**: The resolution limit free method based on the localization approach [Chakraborty *et al.*, 2014]. This method quantifies permanence of a node within a cluster, and it greedily explores clusters so that the permanence increases.
- **TECTONIC**: The motif-aware clustering method proposed by [Tsourakakis *et al.*, 2017]. This method extracts connected components as clusters by removing edges with a smaller number of triangles than threshold θ. We used the recommended value $\theta = 0.06$.

All experiments were conducted on a Linux server with Intel Xeon E5-2690 CPU 2.60 GHz and 128 GB RAM. All algorithms were implemented in C/C++ as a single-threaded program with the entire graph held in the main memory.

We used six real-world graphs published by SNAP [Leskovec and Krevl, 2014] and LAW [Boldi *et al.*, 2011]. Table 1 shows their statistics. The symbols, d and γ denote the average degree and the skewness of the power-law degree distribution, respectively. As shown in Table 1, only YT, LJ, and OK have their ground-truth clusters. In our experiments, we also used synthetic graphs with their ground-truth

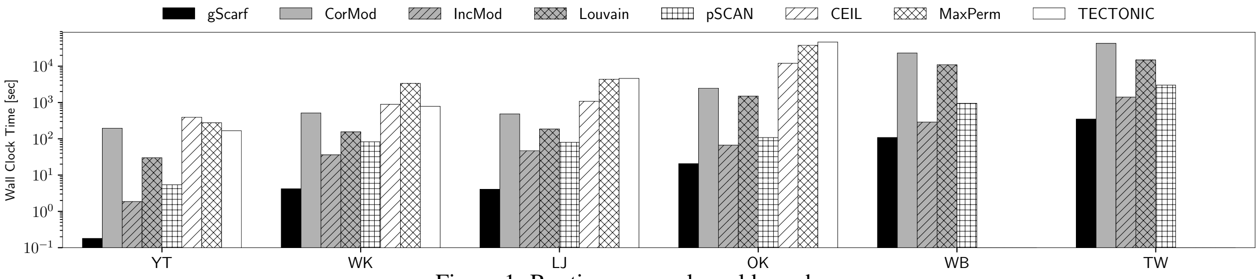


Figure 1: Runtimes on real-world graphs.

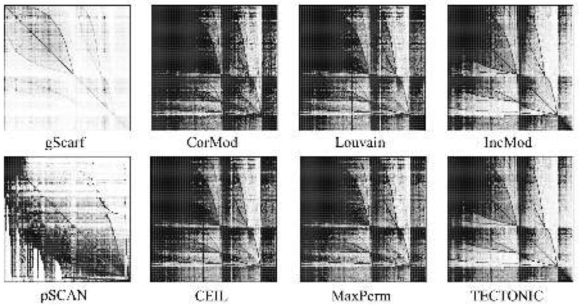


Figure 2: Computed edges on YT in each approach.

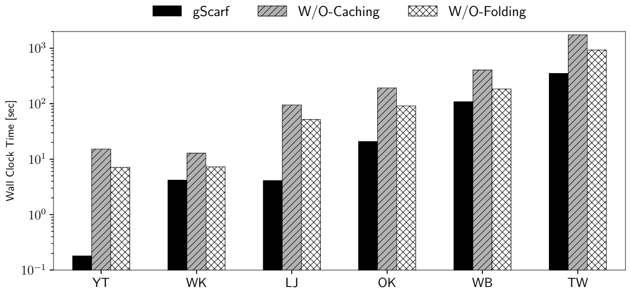


Figure 3: Effectiveness of key approaches.

clusters generated by LFR-benchmark [Lancichinetti *et al.*, 2009], which is the *de facto standard* model for generating graphs. The setting are described in detail in Section 4.2.

4.1 Efficiency

We evaluated the clustering times of each algorithm in the real-world graphs (Figure 1). Note that we omitted the results of CEIL, MaxPerm, and TECTONIC for large graphs since they did not finish within 24 hours. Overall, gScarf outperforms the other algorithms in terms of running time. On average, gScarf is 273.7 times faster than the other methods. Specifically, gScarf is up to 1,100 times faster than CorMod, although they optimize the same clustering criterion. This is because gScarf exploits LRM-gain caching and does not compute all nodes and edges. In addition, it further reduces redundant computations incurred by triangles included in a graph by incrementally folding subgraphs. As a result, as we proved in Theorem 1, gScarf shows a nearly linear running time against the graph size. That is why our approach is superior to the other methods in terms of running time.

To verify how gScarf effectively reduces the number of computations, Figure 2 plots the computed edges in each algorithm for YT where a black (white) dot indicates that the (i,j)-th element is (is not) computed. gScarf decreases the computational costs by 83.9–98.3% compared to the other methods. The results indicate that the LRM-gain caching and the incremental subgraph folding successfully reduce unnecessary computations. This is due to the power-law degree distribution, which is a natural property of real-world graphs including YT. If a graph has a strong skewness in its degree distribution, there are numerous equivalent structural properties since the graph should have similar degrees. Hence, gScarf can remove a large amount of computations.

We then evaluated the effectiveness of our key approaches. We compared the runtimes of gScarf with its variants that exclude either LRM-gain caching or incremental subgraph folding. Figure 3 shows the runtimes of each algorithm in real-world graphs, where *W/O-Caching* and *W/O-Folding* indicate gScarf without LRM-gain caching and incremental subgraph folding, respectively. gScarf is 21.6 and 10.4 times faster than W/O-Caching and W/O-Folding on average, respectively. These results indicate that LRM-gain caching contributes more to the efficiency improvements even though it requires $O(d^{2\gamma})$ times. As discussed in Section 3.5, real-world graphs generally have small values of d and γ due to the power-law degree distribution. For instance, as shown in Table 1, the datasets that we examined have small d and γ, *i.e.,* $d \leq 38.1$ and $\gamma < 2.3$ at most. Consequently, the time complexity $O(d^{2\gamma})$ is much smaller than the cost for incremental subgraph folding in real-world graphs. Hence, gScarf can efficiently reduce its running time.

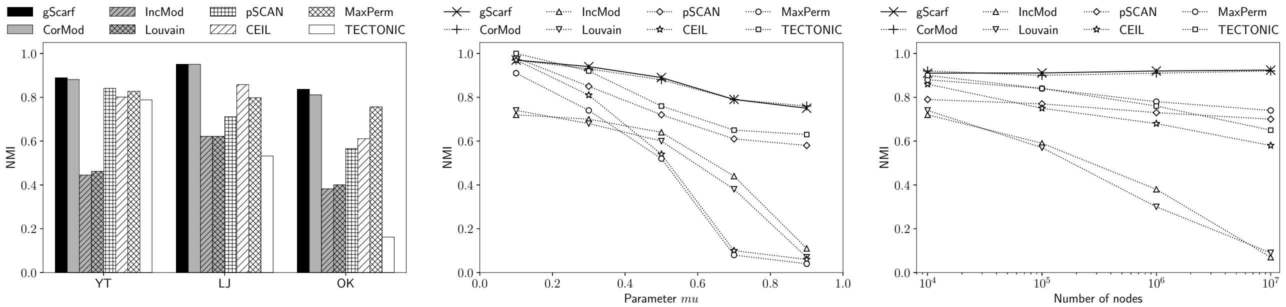
4.2 Accuracy of Clustering Results

One major advantage of gScarf is that it does not sacrifice clustering quality, although it dynamically skips unnecessary computations. To verify the accuracy of gScarf, we evaluated the clustering quality against ground-truth clusters on both real-world and synthetic graphs. We used *normalized mutual information (NMI)* [Cilibrasi and Vitányi, 2005] to measure the accuracy compared with the ground-truth clusters.

Real-world graphs

Figure 4 (a) shows NMI scores of each algorithm in YT, LJ, and OK. We used the top-5000-community datasets [Leskovec and Krevl, 2014] as the ground-truth clusters for all graphs. In the datasets, several nodes belong to multiple ground-truth clusters. We thus assigned such nodes into the cluster where most of its neighbor nodes are located.

As shown in Figure 4, gScarf shows higher NMI scores than the other methods except for CorMod. In addition, gScarf has almost the same or slightly higher NMI score than the original method CorMod. This is because both employ



(a) Real-world graphs (b) LFR benchmark graphs varying *mu*. (c) LFR benchmark graphs varying n.

Figure 4: NMI scores on real-world and synthetic graphs.

	Ground-truth	gScarf	CorMod	Louvain	IncMod	pSCAN	CEIL	MaxPerm	TECTONIC
YT	13.5	13.3	13.3	66.1	50.4	24.3	5.6	11.4	8.2
LJ	40.6	44.2	45.1	111.4	104.5	81.9	11.3	33.3	10.7
OK	215.7	194.9	194.1	16551.9	15676.7	403.7	35.7	121.9	9.61

Table 2: Average cluster sizes.

LRM maximization to avoid the resolution limit problem. Furthermore, gScarf is theoretically designed not to miss any chance to increase LRM by Lemma 1 and Lemma 5. That is, the greedy LRM-gain maximization in gScarf is equivalent to that in CorMod. Hence, gScarf achieves better NMI scores even though it dynamically prunes unnecessary nodes/edges by LRM-gain caching and incremental subgraph folding.

To analyze the cluster resolution, we compared the average sizes of clusters extracted by each algorithm. As shown in Table 2, gScarf and CorMod effectively reproduce the cluster sizes of the ground-truth in large real-world graphs. This is because LRM maximization is designed to avoid the resolution limit problem [Duan *et al.*, 2014], and LRM effectively extract cluster sizes that approximate the ground-truth clusters. On the other hand, other algorithms extract coarse-grained or significantly smaller clusters. Especially in the modularity maximization methods (*i.e.* Louvain and IncMod), they output larger clusters than the ground-truth since they grow clusters until their sizes reach $\sqrt{m}$.

Synthetic graphs

To verify the effects of graph structures and graph sizes, we evaluated the NMI scores on LFR-benchmark graphs. In Figure 4 (b), we generated graphs composed of 10^6 nodes with average degree 20 by varying the mixing parameter *mu* from 0.1 to 0.9. *mu* controls the fraction of neighbor nodes included in other clusters; as *mu* increases, it becomes more difficult to reveal the intrinsic clusters. In contrast, in Figure 4 (c), we used graphs generated by varying the number of nodes from 10^4 to 10^7, where the average degree and *mu* are set to 20 and 0.5, respectively.

Figure 4 (b) shows that gScarf and CorMod achieve high accuracy even if the parameter *mu* increases, whereas most of the other methods degrade NMI scores for large *mu*. In addition, as shown in Figure 4 (c), gScarf also shows higher NMI scores than the others for large graphs since LRM effectively avoids to output super-clusters regardless of inputted graph sizes. Hence, gScarf efficiently detects clusters that effectively approximate the ground-truth clusters in massive graphs. From these results, we confirmed that gScarf does not sacrifice the clustering quality of CorMod.

5 Conclusion

We proposed gScarf, an efficient algorithm that produces fine-grained modularity-based clusters from massive graphs. gScarf avoids unnecessary computations by LRM-gain caching and incremental subgraph folding. Experiments showed that gScarf offers an improved efficiency for massive graphs without sacrificing the clustering quality compared to existing approaches. By providing our efficient approaches that suit for massive graphs, gScarf will enhance the effectiveness of future applications based on the graph analysis.

Acknowledgements

This work was partially supported by JST ACT-I. We thank to Prof. Ken-ichi Kawarabayashi (National Institute of Informatics, Japan) for his helps and useful discussions.

A How to Handle Directed Graphs

We here detail gScarf modifications for directed graphs. As discussed in [Nicosia *et al.*, 2009], we can handle the directed graphs in the modularity maximization by replacing a_i to $a_i^{in} a_i^{out}$ in $ep(i)$, where a_i^{in} and a_i^{out} are in-degree and out-degree of node i, respectively. Thus, our LRM-gain caching can handle the directed graphs by replacing a_i and a_j of $s_{i,j}$ in Definition 2 to $a_i^{in} a_i^{out}$ and $a_j^{in} a_j^{out}$, respectively. Similarly, we need to modify Definition 3 so that the folding technique takes account of directions of edges. That is, our incremental subgraph folding can handle the directed graphs by aggregating in-degree edges and out-degree separately.

References

[Blondel *et al.*, 2008] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E.L.J.S. Mech. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[Boldi *et al.*, 2011] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proc. WWW 2011*, pages 587–596, 2011.

[Brin *et al.*, 1997] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *ACM SIGMOD Record*, 26(2):255–264, 1997.

[Chakraborty *et al.*, 2014] Tanmoy Chakraborty, Sriram Srinivasan, Niloy Ganguly, Animesh Mukherjee, and Sanjukta Bhowmick. On the Permanence of Vertices in Network Communities. In *Proc. KDD 2014*, pages 1396–1405, 2014.

[Chang *et al.*, 2017] Lijun Chang, Wei Li, Lu Qin, Wenjie Zhang, and Shiyu Yang. pSCAN: Fast and Exact Structural Graph Clustering. *IEEE Transaction on Knowledge Data Engineering*, 29(2):387–401, 2017.

[Cilibrasi and Vitányi, 2005] Rudi Cilibrasi and Paul MB Vitányi. Clustering by Compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[Clauset *et al.*, 2004] Aaron Clauset, M. E. J. Newman, , and Cristopher Moore. Finding Community Structure in Very Large Networks. *Physical Review*, E 70(066111), 2004.

[Costa, 2014] Alberto Costa. Comment on "Quantitative Function for Community Detection". *CoRR*, abs/1409.4063, 2014.

[Duan *et al.*, 2014] Lian Duan, Willian Nick Street, Yanchi Liu, and Haibing Lu. Community Detection in Graphs Through Correlation. In *Proc. KDD 2014*, pages 1376–1385, 2014.

[Faloutsos *et al.*, 1999] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. In *Proc. SIGCOMM 1999*, pages 251–262, 1999.

[Fortunato and Barthélemy, 2007] Santo Fortunato and M Barthélemy. Resolution Limit in Community Detection. *Proceedings of the National Academy of Sciences of United States of America*, 104(1):36–41, Jan 2007.

[Lancichinetti *et al.*, 2009] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the Overlapping and Hierarchical Community Structure in Complex Networks. *New Journal of Physics*, 11(3):033015, 2009.

[Leskovec and Krevl, 2014] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, jun 2014.

[Li *et al.*, 2008] Zhenping Li, Shihua Zhang, Rui-Sheng Wang, Xiang-Sun Zhang, and Luonan Chen. Quantative Function for Community Detection. *Physical Review*, E 77(036109), 2008.

[Louni and Subbalakshmi, 2018] Alireza Louni and K. P. Subbalakshmi. Who Spread That Rumor: Finding the Source of Information in Large Online Social Networks with Probabilistically Varying Internode Relationship Strengths. *IEEE Transactions on Computational Social Systems*, 5(2):335–343, June 2018.

[Muff *et al.*, 2005] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. Local Modularity Measure for Network Clusterizations. *Physical Review*, E 72(056107), 2005.

[Newman, 2004] M. E. J. Newman. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review*, E 69(066133), 2004.

[Nicosia *et al.*, 2009] Vincenzo Nicosia, Giuseppe Mangioni, Vincenza Carchiolo, and Michele Malgeri. Extending the Definition of Modularity to Directed Graphs with Overlapping Communities. *Journal of Statistical Mechanics*, 2009(3):P03024, 2009.

[Palla *et al.*, 2005] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. *Nature*, 435(7043):814–818, June 2005.

[Papoulis and Pillai, 2002] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Higher Education, 4 edition, 2002.

[Piatetsky-Shapiro, 1991] Gregory Piatetsky-Shapiro. Discovery, Analysis, and Presentation of Strong Rules. *Knowledge Discovery in Databases*, pages 229–248, 1991.

[Sankar *et al.*, 2015] M. Vishnu Sankar, Balaraman Ravindran, and S Shivashankar. CEIL: A Scalable, Resolution Limit Free Approach for Detecting Communities in Large Networks. In *Proc. IJCAI 2015*, pages 2097–2103, 2015.

[Sato *et al.*, 2018] Tomoki Sato, Hiroaki Shiokawa, Yuto Yamaguchi, and Hiroyuki Kitagawa. FORank: Fast ObjectRank for Large Heterogeneous Graphs. In *Companion Proceedings of The Web Conference 2018*, pages 103–104, 2018.

[Shiokawa *et al.*, 2013] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *Proc. AAAI 2013*, pages 1170–1176, 2013.

[Shiokawa *et al.*, 2015] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 8(11):1178–1189, July 2015.

[Shiokawa *et al.*, 2018] Hiroaki Shiokawa, Tomokatsu Takahashi, and Hiroyuki Kitagawa. ScaleSCAN: Scalable Density-based Graph Clustering. In *Proc. DEXA 2018*, pages 18–34, 2018.

[Takahashi *et al.*, 2017] Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors. In *Proc. ACM SIGMOD Workshop on Network Data Analytics*, pages 6:1–6:7, 2017.

[Tsourakakis *et al.*, 2017] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable Motif-aware Graph Clustering. In *Proc. WWW 2017*, pages 1451–1460, 2017.

[Watts and Strogatz, 1998] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393(6684):440–442, 1998.