Journal of Data Intelligence, Vol. 1, No. 2 (2020) 101–123

EFFICIENT VECTOR PARTITIONING ALGORITHMS FOR MODULARITY-BASED GRAPH CLUSTERING

HIROAKI SHIOKAWA
Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
shiokawa@cs.tsukuba.ac.jp

YASUNORI FUTAMURA
Faculty of Engineering, Information and Systems, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
futamura@cs.tsukuba.ac.jp

This paper addressed the problem of finding clusters included in graph-structured data such as Web graphs, social networks, and others. Graph clustering is one of the fundamental techniques for understanding structures present in the complex graphs such as Web pages, social networks, and others. In the Web and data mining communities, the modularity-based graph clustering algorithm is successfully used in many applications. However, it is difficult for the modularity-based methods to find fine-grained clusters hidden in large-scale graphs; the methods fail to reproduce the ground truth. In this paper, we present a novel modularity-based algorithm, *CAV*, that shows better clustering results than the traditional algorithm. The proposed algorithm employs a cohesiveness-aware vector partitioning into the graph spectral analysis to improve the clustering accuracy. Additionally, this paper also presents a novel efficient algorithm *P-CAV* for further improving the clustering speed of CAV; P-CAV is an extension of CAV that utilizes the thread-based parallelization on a many-core CPU. Our extensive experiments on synthetic and public datasets demonstrate the performance superiority of our approaches over the state-of-the-art approaches.

Keywords: Graph, Clustering, Modularity

1. Introduction

Graph is a fundamental data structure that represents schema-less and complecated relationships among data entities [1]. Due to recent advances in Web-based applications and systems, large-scale graphs are now ubiquitous; for instance, social networks (*e.g.,* Facebook and Twitter) are the representative ones. To understand such large-scale graphs, graph clustering [2, 3, 4] (*a.k.a.* community detection) is one of the most important techniques in various research areas such as social network analysis [5] and Web sciences [6]. The graph clustering divides a graph into several groups of nodes, so-called clusters, whose nodes are highly connected inside. By using graph clustering, we can discover the structures and representative examples hidden in the given graph.

The problem of finding clusters in a graph has been studied for some decades in database and Web communities. Graph partitioning algorithms [7, 8] and supervised approaches [9] are the natural choices for this problem. Since graph structures in the real-world applications are highly complex, *modularity-based algorithm*, proposed by Newman and Girvan [10], at-

tracts a great deal of attention for extracting such clusters. The modularity-based algorithm detects a set of clusters that maximize a clustering metric *modularity* [10]. The modularity evaluates the fraction of edges inside clusters as compared to edges between clusters; the better clustering results are achieved with higher modularity scores. By finding clusters that maximize the modularity, we can extract reasonable groups of nodes from a given graph. Due to the effectiveness of the modularity, the modularity-based algorithms have been applied in many applications including Web-based applications and systems such as event detection on SNSs [11], route reccomendation [12], brain analysis [13] and so on.

Although the modularity is useful for many applications, it has a serious problem, so-called *resolution limit problem* [14]. Recent research pointed out that the modularity measure is not a scale-invariant measure [15], and thus it is difficult for modularity maximization methods to find fine-grained (small) clusters hidden in large graphs. Since the sizes of graphs are recently increasing [16], the resolution limit problem causes a critical deterioration in the quality of Web-based applications and systems [17].

To address the problem, several algorithms have been proposed in the physics community. Since the modularity implicitly assumes that each node can interact with every other node, it is more reasonable for real-world graphs to assume that each node interacts just with its local neighborhood nodes [18]. Based on the above observation, Muff *et al.,* and Li *et al.,* first proposed *localized modularity measure* [18] and *modularity density measure* [5], respectively. Unlike the original modularity [10], these methods compute the fraction of edges within a cluster *w.r.t.* the subgraph including the cluster and its neighbor clusters. Although these approaches certainly succeeded in avoiding the resolution limit problem for the small graphs (*e.g.,* less than 10^2 nodes), Costa [19] recently pointed out that they are still affected by the resolution limit problem for large graphs. That is, the methods tend to produce only large clusters if a given graph is large. For further improving the clustering quality on large graphs, several algorithms, *i.e., SFO* [20] and *ZN* [21], recently proposed; however, they also suffer from the resolution limit of the modularity. Hence, it is still a challenging task to capture fine-grained clusters from large-scale graphs.

1.1. *Our approach and contributions*

We focus on the problem above to efficiently find highly accurate clusters included in large-scale graphs. In this paper, we present a novel algorithm, namely *cohesiveness-aware vector partitioning CAV* [22] to avoid the resolution limit problem of the modularity. Our proposed approach CAV is designed to capture *cohesiveness* of each cluster during the modularity maximization procedure; it extracts sets of nodes as clusters such that each of which shows (1) high modularity and (2) high cohesiveness within a cluster simultaneously. As a result, our proposed method CAV detects more accurate clusters than those of traditional modularity maximization algorithms [20, 21].

Although CAV is capable of detecting fine-grained clusters, it requires expensive computational costs to compute large-scale graphs [22]. This is because that CAV needs to compare the all pairs of nodes to evaluate a cohesiveness for each node during the clustering procedure. This exhaustive procedure entails $O(n^3)$ times to obtain a clustering result from a given graph, where n is many nodes in the graph. Thus, it is still difficult for CAV to handle large-scale graphs.

For further improving our previous approach, we additionally propose an extension of CAV, namely *P-CAV*, as a parallel algorithm for the cohesiveness-aware vector partitioning. As we described earlier, our previous approach CAV needs to perform exhaustive computations to evaluate the cohesiveness. To reduce such expensive computational costs, we fully utilize multiple physical cores equipped on a modern CPU chip. That is, our proposed algorithm P-CAV is a thread-parallel algorithm that performs on a many-core CPU. By parallelizing the exhaustive cohesiveness computations on a many-core CPU, we attempt to achieve improvement of clustering speeds.

As a result, our proposed algorithms CAV and P-CAV have the following attractive characteristics:

- **Accurate:** Our cohesiveness-aware vector partitioning CAV is considerably more accurate than the state-of-the-art algorithms [20, 21]. Moreover, our extension algorithm P-CAV always returns the same clustering results as those of CAV [22], which returns highly accurate clustering results. That is, P-CAV does not sacrifice the high clustering accuracy of CAV, and it successfully moderates the resolution limit effect of the modularity-based graph clustering.
- **Efficient:** Compared with our naïve approach CAV, P-CAV achieves higher speed clustering by using the above approach for cohesiveness computations; P-CAV successfully moderates the expensive costs of the cohesiveness computations. Furthermore, our proposed algorithm P-CAV shows near-linear speeding up as increasing of the number of threads. P-CAV is also scalable to the sizes of graphs.
- **Easy to deploy:** Our proposed methods CAV and P-CAV do not require the number of clusters before performing the cluster analysis. This property allows users to deploy the graph clustering algorithm into real-world applications easily.

Our extensive experiments showed that P-CAV runs at most 27 times faster than CAV without sacrificing the clustering quality. Also, P-CAV and CAV achieve 60% higher accuracy in the best cases than those of the state-of-the-art modularity-based methods [20, 21]. These characteristics and results confirm the practicality of our algorithm for real-world applications. Even though the modularity-based algorithms are useful in enhancing application quality, it has been difficult to apply the algorithms to real-world applications due to their resolution limit problem. However, by providing our fast and highly accurate approaches that suit the identification of clusters on large-scale graphs, our proposed algorithms CAV and P-CAV will help to improve the effectiveness of a wider range of applications.

The rest of this paper is organized as follows: Section 2 describes a brief background of this work. Section 3 and 4 introduce our proposed approaches CAV and P-CAV, respectively. After that, we report the experimental results in Section 5. In Section 6, we briefly review the related work, and we conclude this paper in Section 7.

2. Preliminary

In this section, we formally define the notations and introduce the background of this paper. Let $G = (V, E)$ be an unweighted and undirected graph, where V and E are a set of nodes and edges, respectively. We assume that graphs are undirected and unweighted only to simplify

Table 1. Definitions of main symbols.

Notation	Definition
G	A given graph
V	A set of nodes in G
E	A set of edges in G
$\mathbb{C}$	A set of clusters
$\mathbb{O}$	A set of outliers
n	The number of nodes, *i.e.*, $n = \|V\|$
m	The number of edges, *i.e.*, $m = \|E\|$
ω	A user-specified parameter, $\omega \in [0, 180]$
μ	A user-specified parameter, $\mu \in \mathbb{N}$ and $\mu \geq 2$
$\mathbf{A}$	An adjacency matrix of G
$\mathbf{B}$	An modularity matrix of G
λ_l	l-th eigenvalue of $\mathbf{B}$
$\mathbf{U}$	A matrix whose l-th column is the eigenvalues of λ_l
d_i	Degree of node i
$\mathbf{r}_i$	A node-vector of node i defined by Definition 2
N_i^{ω}	ω-neighborhood of node i given by Definition 3
P_i	A cluster-partition of node i given by Definition 5

the representations. Other types of graphs such as directed and weighted, can be handled with only slight modifications. Table 1 lists the main symbols and their definitions.

2.1. *Modularity*

Modularity, introduced by Newman and Girvan, is widely used to evaluate the cluster structure of a given graph from global perspective [10]. Modularity is a quality metric of graph clustering that measures the differences in graph structures from an expected random graph. The main idea of modularity is to find a group of nodes that have a lot of inner-group edges and few inter-group edges. Formally, modularity Q is defined as follows:

Definition 1 (Modularity Q) *Let g_i be a cluster to which node i belongs, the modularity Q is defined as*

$$Q = \frac{1}{2m} \sum_{ij} \left\{ A_{ij} - \frac{d_i d_j}{2m} \right\} \delta_{g_i, g_j}, \tag{1}$$

where δ_{g_i,g_j} is a Kronecker delta. It returns 1 if $g_i = g_j$; otherwise, 0.

As we can see from Definition 1, the modularity Q is a sum of subtractions $\left\{ A_{ij} - \frac{d_i d_j}{2m} \right\} \delta_{g_i,g_j}$ for all pairs of clusters such that $g_i = g_j$. In each subtraction, A_{ij} represents the fraction of edges included in a cluster g_i (or g_j). On the other hand, $\frac{d_i d_j}{2m}$ is the expected fraction of edges when we assume the graph to be a random graph. Thus, the modularity Q increases if a clustering result has a larger fraction of edges for each cluster than those of a random graph. That is, well-clustered graphs will have large modularity scores, and optimal clustering is achieved when modularity is maximized. In this paper, we denote $Q(C)$ to represent a contribution of cluster C for the modularity Q.

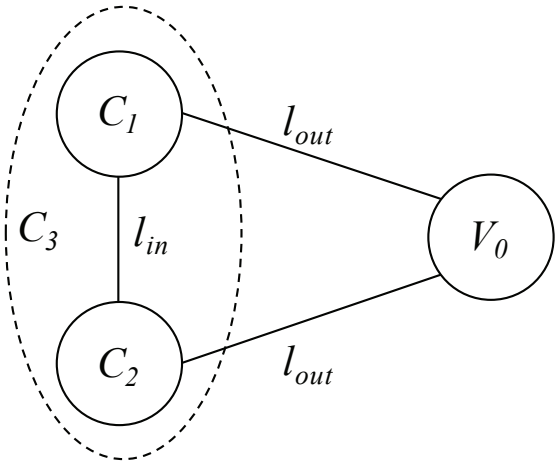

Fig. 1. Example graph: The graph in Figure 1 is consist of three groups, C_1, C_2,and V_0; C_1 and $C_2$are clusters extracted by a modularity-based clustering, and V_0 represents the rest of $C_1$and C_2 in the graph.

2.2. *Resolution limit problem of modularity*

Based on Definition 1, modularity-based graph clustering algorithms [20, 21, 23] explore a set of clusters that optimizes the modularity Q. However, despite the efficiency of the approach, recent research pointed out that modularity is not a scale-invariant measure. Thus, it is difficult for modularity maximization methods to find small clusters hidden in large-scale graphs; that is, these methods fail to fully reproduce the ground-truth. This serious weakness of modularity Q is famously known as the *resolution limit problem* [14]. In this section, we here theoretically discuss how the modularity Q fails to find fine-grained clusters on graphs.

In order to simplify the discussion, we suppose that we have a graph shown in Figure 1. The graph in Figure 1 is consisted of three groups, C_1, C_2, and V_0. C_1 and C_2 are clusters such that $Q(C_1) > 0$ and $Q(C_2) > 0$, respectively; V_0 represents the rest of the graph. As shown in Figure 1, C_1 and C_2 have l internal edges within each of them, and C_1 (or C_2) are connected by l_{in} and l_{out} edges with C_2 (or C_1) and V_0, respectively.

Let us consider to construct C_3 by merging C_1 and C_2, and we have a gain of modularity Q as $Q(C_3)-Q(C_1)-Q(C_2)$. Now, we expect C_1 and C_2 should be separated by the modularity Q optimization, *i.e.,* $Q(C_3) < Q(C_1) + Q(C_2)$. In this case, from Definition 1, we should have the following condition:

$$l > \frac{2ma}{(a+b+2)^2}, \tag{2}$$

where $a = \frac{l_{in}}{l}$ and $b = \frac{l_{out}}{l}$. Since $a + b < 2$ if $Q(C_1) > 0$ (or $Q(C_2) > 0$) from the literature [24], we finally have the following condition by rearranging Inequation (2):

$$l > \sqrt{\frac{m}{2}} - 1 \approx \sqrt{\frac{m}{2}}. \tag{3}$$

This condition means that C_1 and C_2 should be merged into C_3 until the number of edges

included in a cluster C_1 (or C_2) exceeds $\sqrt{\frac{m}{2}}$ even though $\sqrt{\frac{m}{2}}$ edges are the greater part of E. As Fortunato *et al.,* proved that the above resolution limit problem is observed in various real-world graphs, and in some extreme cases, the modularity maximization grows $l \approx \sqrt{2m}$. From the above reasons, the modularity-based algorithm fails to extract fine-grained clusters from large graphs that have large m.

3. Naïve Algorithm: (CAV)

Our goal is to efficiently detect fine-grained clusters from a given graph so as to avoid the resolution limit. In this section, we introduce our naïve algorithm, named *CAV* (Cohesiveness-aware Vector Partitioning) [22], that is designed to moderate the resolution limit effect for the modularity-based graph clustering. We first overview the ideas underlying CAV and then give a full description of the graph clustering algorithm.

3.1. *Overview of CAV*

The basic idea underlying CAV is to capture the cohesiveness of each cluster during the modularity Q maximization. If nodes have enough cohesiveness, our proposed method groups them into the same cluster; otherwise, it excludes the nodes from clusters. As discussed in Section 2, the modularity Q has a serious weakness; the modularity Q merges two nodes into the same cluster even though the nodes have sparse connections between them. This weakness entails clusters with low cohesiveness, especially for large graphs, since the clusters contain nodes as members of them regardless of cohesiveness. Thus, by integrating the cohesiveness into the original modularity maximization techniques, CAV attempts to improve the clustering quality.

To avoid the resolution limit, CAV employs the following two approaches:

(1) Node-vector extraction: CAV first constructs a node-vector for each node in the given graph by performing the spectral analysis on the modularity Q. CAV rewrites Definition 1, and transforms the modularity maximization problem into the node-vector partitioning problem.

(2) Cohesiveness-aware vector partitioning: We now have a node-vector representation for each node i through the spectral analysis on the modularity Q. Thus, in this phase, we design the cluster detection as the problem of finding a division of the node vectors so that it shows better clustering accuracy than the traditional modularity-based algorithm. In order to enhance the clustering quality, we introduce a heuristic vector partitioning algorithm that is based on the cohesiveness of node-vectors. By performing the node-vector partitioning, CAV detects nodes with dense edge connections as a cluster and excludes sparsely connected nodes as outliers.

These simple ideas have three main advantages:

First, CAV can output accurate clusters. Our ideas successfully avoid the resolution limit of modularity-based graph clustering by introducing the cohesiveness into the vector partitioning problem. As we described in Section 2, the modularity Q increases when two adjacent nodes are merged into the same cluster even if the edge connections are sparse between the two nodes. In contrast, our proposed method excludes such sparsely connected nodes from the clusters; thus it can achieve more cohesive clustering results than the traditional modularity maximization. In Section 5.3, we experimentally confirm that how CAV successfully achieves

highly accurate clustering results for large graphs.

Second, CAV outputs robust clusters. This is because that our proposed method extracts unique clusters once the parameters and a graph are determined while the spectral algorithms do not ensure uniqueness of the clusters. We theoretically discuss the robustness of our proposal in Theorem 1.

Last, CAV can detect all clusters by traversing all nodes only once even though it does not require the number of clusters before performing the clustering procedure. Due to the uniqueness of the clustering results, our proposal can extract all clusters from the given graph without iteratively computing all nodes as similar to the traditional spectral algorithms (Theorem 2). Furthermore, it automatically determines the number of clusters included in the given graph. Thus our proposed method can achieve better clustering results compared with the spectral algorithms.

3.2. *Node-vectors extraction*

We first construct a p-dimensional node-vector [21] for each node from Definition 1 through spectral analysis. For ease of the analysis, we introduce a symmetric $n \times n$ matrix $\mathbf{B}$ whose element B_{ij} is defined by the following equation:

$$B_{ij} = A_{ij} - \frac{d_i d_j}{2m}. \tag{4}$$

Clearly, the matrix $\mathbf{B}$ is symmetric, thus we can rewrite Definition 1 by applying the eigenvector decomposition to $\mathbf{B}$. Specifically, we hold the following equations:

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{ij} \sum_{l=1}^{n} \lambda_l U_{il} U_{jl} \delta_{g_i, g_j} && (5) \\ &= \frac{1}{2m} \sum_{ij} \sum_{l=1}^{n} \lambda_l U_{il} U_{jl} \sum_{s} \delta_{s,g_i} \delta_{s,g_j} && (6) \\ &= \frac{1}{2m} \sum_{l=1}^{n} \lambda_l \sum_{s} \left[\sum_{i} U_{il} \delta_{s,g_i} \right]^2, && (7) \end{aligned}$$

where λ_l is an eigenvalue of $\mathbf{B}$ and U_{il} is an element of the orthogonal matrix $\mathbf{U}$ whose columns are the corresponding eigenvectors. In Equation (5), we assume that the eigenvalues are numbered in decreasing order, *i.e.* $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

As we can see from Equation (5), the modularity Q is a sum over eigenvalues λ_l times the non-negative quantities $\sum_s \left[\sum_i U_{il} \delta_{s,g_i}\right]^2$. Thus, we can obtain the largest modularity Q by summing only the terms corresponding to the positive eigenvalues since the negative eigenvalues never increase the modularity Q shown in Equation (5). That is, we can approximate Equation (5) by the following equation.

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{l=1}^{p} \lambda_l \sum_{s} \left[\sum_{i} U_{il} \delta_{s,g_i} \right]^2 && (8) \\ &= \frac{1}{2m} \sum_{s=1}^{k} \sum_{l=1}^{p} \left[\sum_{i} \sqrt{\lambda_I} U_{il} \delta_{s,g_i} \right]^2, && (9) \end{aligned}$$

where s represents a cluster, and an integer value p is the number of positive eigenvalues. From Equation (8), we finally define a p-dimensional node-vector for each node as follows:

Definition 2 (Node-vector) *Let $\boldsymbol{r}_i \in \mathbb{R}^p$ be a p-dimensional node-vector of node i, the l-th element ($1 \leq l \leq p$) of $\boldsymbol{r}_i$ is as follows:*

$$r_i[l] = \sqrt{\lambda_l} U_{il}. \tag{10}$$

Note that, from Definition 2, we finally rewrite the modularity Q as follows:

$$Q \approx \frac{1}{2m} \sum_{s=l}^{k} \sum_{l=1}^{p} \left(\sum_{i \in s} r_i[l] \right)^2 = \frac{1}{2m} \sum_{s=1}^{k} \left| \sum_{i \in s} \mathbf{r}_i \right|^2, \tag{11}$$

where the notation $i \in s$ denotes the node i is in cluster s. From the above equation, we can consider the modularity-based clustering as the problem of finding k clusters that maximize Equation (11).

3.3. *Cohesiveness-aware vector partitioning*

We detect clusters included in the given graph by using the set of node-vectors that we have derived in the previous section. As we described in Section 3.2, we can approximately maximize the modularity Q by finding a division of the set of node-vectors. This is because that the modularity Q never takes negative values for any partitions of the node-vectors since, as shown in Equation (11), the modularity Q is a sum of non-negative quantities $\left|\sum_{i \in s} \mathbf{r}_i\right|^2$. Thus, it is essential to find a division of the node-vectors so that the partition avoids the resolution-limit problem [14] of the modularity Q.

To detect a proper partition of node-vectors, we present a heuristic algorithm, named *cohesiveness-aware vector partitioning*, that focuses on the *cohesiveness* among the node-vectors. Generally, traditional spectral algorithms [7, 21] partition the node-vectors into k-disjoint groups without regarding the cohesiveness of node-vectors included in each partition. In contrast, our approach extracts only groups of node-vectors as clusters if the node-vectors have enough cohesiveness; otherwise, it excludes the node-vectors from clusters. In order to evaluate the cohesiveness between the two node-vectors, we first introduce the following definition.

Definition 3 (ω-neighborhood) *Given a user-specified parameter $\omega \in [0, 180]$, ω-neighborhood of node i, denoted by N_i^ω, is a set of nodes defined as follows:*

$$N_i^\omega = \left\{ j \in V \;\middle|\; \frac{180}{\pi} \arccos\left(\frac{\boldsymbol{r}_i^T \boldsymbol{r}_j}{|\boldsymbol{r}_i||\boldsymbol{r}_j|} \right) \leq \omega \right\}. \tag{12}$$

Definition 3 indicates that N_i^ω is a set of node-vectors, each of which has a vectorial angle less than ω with node-vector $\mathbf{r}_i$.

From Definition 3, our approach then detects *core-vector*, which plays a seed of a partition construction, by finding ω-neighborhood for all node-vectors. In order to specify core-vector metrics, we introduce a user-specified parameter μ; it is a parameter for controlling the

minimum size of a partition. Our approach regards a node-vector $\mathbf{r}_i$ as a core-vector when it has $|N_i^\omega| \geq \mu$.

Definition 4 (Core-Vector) *Given a minimum partition size $\mu \in \mathbb{N}$ s.t. $\mu \geq 2$, node-vector $\boldsymbol{r}_i$ is a core-vector iff $|N_i^\omega| \geq \mu$.*

Once our approach finds a core-vector $\mathbf{r}_i$, it expands a partition (cluster) from the core-vector $\mathbf{r}_i$. Specifically, the node-vectors included in N_i^ω are assigned to the same partition as the core-vector $\mathbf{r}_i$. If a node-vector $\mathbf{r}_j \in N_i^\omega$ is also a core-vector, our approach again expands the partition from $\mathbf{r}_j$; Finally, it stops expanding the partition from $\mathbf{r}_j$ when the partition has no unvisited core-vectors. As a result, our approach detects the following cluster-partition, which represents a cluster of $\mathbf{r}_i$, defined as follows:

Definition 5 (Cluster-Partition) *Let P_i be a cluster-partition what a node i belongs, the cluster-partition P_i is given by the following equation.*

$$P_i = \{w \in N_j^\omega | j \in P_i \wedge |N_j^\omega| \geq \mu\}, \tag{13}$$

where the cluster-partition P_i is initially set to $P_i = \{i\}$.
Our approach continues the above partition expansion until no core-vectors can be found in the given graph.

After terminating the partition expansions, we can identify *non-partitioned* node-vectors that do not belong to any cluster-partitions defined in Definition 5. From Definition 3, 4, and 5, the non-partitioned node-vectors clearly have weaker cohesiveness than the node-vectors in the cluster-partitions since they are not included in any ω-neighborhood. That implies that the nodes, which correspond to the non-partitioned node-vectors, have sparse connections with the other nodes. Thus, in this paper, we regard the non-partitioned node-vectors as *outlier nodes*, and our approach assigns a singleton partition for each non-partitioned node-vector.

Finally, our approach returns the following clustering results:

Definition 6 (Clusters) *Let k be the number of cluster-partitions, the a set of clusters, denoted by $\mathbb{C}$, is given as follows:*

$$\mathbb{C} = \{P_1, P_2, \ldots, P_k \mid |P_i| \geq 2\}. \tag{14}$$

That is, from Definition 6, our approach returns only cluster-partitions as clusters while it excludes the singleton partitions, which contain a non-partitioned node-vector, from the clustering result.

3.4. *Algorithm*

Our algorithm CAV enhances the clustering results of the traditional modularity maximization methods by introducing the cohesiveness of node-vectors in each cluster. The pseudocode of CAV is shown in Algorithm 1. Algorithm 1 takes undirected and unweighted graph G and two user-specified parameters, ω and μ, and it outputs a set of clusters $\mathbb{C}$ and outliers $\mathbb{O}$. The main part of CAV is twofold: (1) extracting node-vectors given by Section 3.2 (lines 1-13), and (2) cohesiveness-aware vector partitioning given by Section 3.3 (lines 14-31).

(1) Extracting node-vectors (lines 1-13): The goal of this phase is to construct a node-vector defined in Definition 2 for each node. First, CAV applies the eigenvalue decomposition to a matrix $\mathbf{B}$ given by Equation (4), and it obtains all eigenvalues and corresponding eigenvectors (lines 1-6). Then, it finds p eigenvalues that are greater than 0 (line 7). Finally, our method construcs the node-vector $\mathbf{r}_i$ for each node i by using p eigenvalues and corresponding eigenvectors by following Definition 2 (lines 8-13).

(2) Cohesiveness-aware vector partitioning (lines 14-31): In this phase, CAV outputs a set of clusters and outliers, denoted by $\mathbb{C}$ and $\mathbb{O}$, respectively. CAV first initializes $\mathbb{C}$ and $\mathbb{O}$ as $\emptyset$ (line 14), and then it starts to detect cluster-partitions. In this cluster-partition detection procedure shown in (line 15-25), CAV performs the following steps until it finds all cluster-partitions included in G: **(a)** It makes initial cluster-partition $P_i = \{i\}$ (line 16). **(b)** It checks node-vector $\mathbf{r}_i$ whether it is a core-vector (*i.e.,* $|N_i^\omega| \geq \mu$) or not by following Definition 4 (lines 18-19). **(c)** It expands the partition P_i if $\mathbf{r}_i$ is a core-vector (lines 19-22). **(d)** It continues **(b)** and **(c)** if unvisited nodes are included in P_i. **(e)** It stores P_i into $\mathbb{C}$ if P_i satisfies Definition 6 (lines 24-26), otherwise, it removes P_i. After termination of the above cluster detection, CAV assigned a singleton-partition for each node corresponding to a non-partitioned node-vector (lines 28-29). It saves the nodes into a set of outliers $\mathbb{O}$ (line 30), and CAV finally outputs a set of clusters $\mathbb{C}$ and outliers $\mathbb{O}$.

As a result, our cohesiveness-aware vector partitioning approach has the following attractive properties:

Theorem 1 *Let P_i be a cluster-partition included in $\mathbb{C}$, which are produced by CAV, P_i is uniquely determined by the given graph G, and two user-specified parameters, ω and μ.*

Proof: From Definition 5, each cluster-partition has at least one core-vector. If the number of core-vectors included in P_i is equal to 1, Theorem 1 clearly holds since $P_i = N_i^\omega$. Hence, we discuss whether Theorem 1 also holds when P_i has multiple core-vectors. As shown in Definition 5, a core-vector $\mathbf{r}_j$ should have at least one another core-vector $\mathbf{r}_k$ in P_i since any of two core-vectors, say $\mathbf{r}_j$ and $\mathbf{r}_k$, can not be in the same cluster-partition by Definition 5 if both $j \notin N_k^\omega$ and $k \notin N_j^\omega$ hold. This deduction implies that we can reach all core-vectors included in P_i from an arbitrary core-vector by traversing the inclusion relations between nodes corresponding core-vectors and ω-neighborhood of them. Thus, in the case of multiple core-vectors, Theorem 1 holds since all core-vectors are reachable and $P_i = \bigcup_{j \in C_i} N_j^\omega$ by Definition 5, where C_j is a set of core-vectors included in P_i. □

Theorem 2 *CAV can find all cluster-partitions by traversing all node-vectors only once.*

Proof: As we proved in Theorem 1, all core-vectors in the same cluster-partition are reachable by tracing the inclusion relations. That is, CAV can detect exactly same cluster-partitions by any ordering of the traversing node-vectors, so once a node-vector traversed, we do not need to compute the node-vector again. Therefore, CAV can find all cluster-partitions by traversing all node-vectors only once. □

4. Parallel Algorithm: Parallel Cohesiveness-aware Vector Partitioning (P-CAV)

Algorithm 1 Naïve Algorithm: CAV

Input: A graph $G = (V, E)$, and parameters, ω and μ;
Output: A set of clusters $\mathbb{C}$ and a set of outliers $\mathbb{O}$;

▷ **(1) Extracting node-vectors**

Obtain $\mathbf{B}$ by Equation (4);
for each $i \in V$ **do**
    for each $j \in V$ **do**
        Get $B_{ij} = \sum_{l=1}^{n} \lambda_l U_{il} U_{jl}$ by eigenvalue decomposition;
Get $p = |\{\lambda_1, \lambda_2, \ldots, \lambda_n \mid \lambda_i > 0, 1 \leq i \leq n\}|$;
for each $i \in V$ **do**
    for $l = 1$ to p **do**
        $r_i[l] = \sqrt{\lambda_l} U_{il}$ from Definition 2;
    Obtain $\mathbf{r}_i = (r_i[1], r_i[2], \ldots, r_i[p])^T$;

▷ **(2) Cohesiveness-aware vector partitioning**

$\mathbb{C} = \emptyset$, $\mathbb{O} = \emptyset$;
for each unvisited node $i \in V$ **do**
    $P_i = \{i\}$, and Mark i as visited;
    for each unvisited node $j \in P_i$ **do**
        Obtain N_j^{ω} by Definition 4;
        if $|N_j^{\omega}| \geq \mu$ **then**
            $P_i = P_i \cup N_j^{\omega}$;
        Mark j as visited;
    if $|P_i| \geq 2$ **then**
        $\mathbb{C} = \mathbb{C} \cup \{P_i\}$;
for each non-partitioned node-vector $\mathbf{r}_i$ **do**
    $P_i = \{i\}$;
    $\mathbb{O} = \mathbb{O} \cup \{P_i\}$;
return $\mathbb{C}$ and $\mathbb{O}$;

In this section, we present a parallelized algorithm for the cohesiveness-aware vector partitioning (P-CAV), which is an extension of CAV [22].

4.1. *Overview of P-CAV*

The major drawback of CAV is its expensive computational cost for large-scale graphs. That is, as shown in Algorithm 1, CAV needs to compute all pair of node vectors to evaluate the cohesiveness of each node; this computation entails $O(n^3)$ times for the clustering. Thus, in this section, we present an extended approach of CAV, named P-CAV (Parallel Cohesiveness-aware Vector Partitioning).

The basic idea underlying P-CAV is to reduce the computational cost for the cohesiveness computations from algorithmic and parallel processing perspectives. Specifically, we employ a thread-parallel node-pruning method that drops unnecessary and redundant computations from the cohesiveness computations. By avoiding such unnecessary and redundant computations, P-CAV attempts to mitigate the expensive cost for the cohesiveness-aware vector partitioning.

Algorithm 2 shows the pseudocode of P-CAV. Differ from Algorithm 1, P-CAV employs parallelized algorithms for detecting clusters from a set of node-vectors (lines 14-32). Once P-CAV obtains all node-vectors from a given graph, it first detects all core-vectors by performing thread-parallel node pruning (lines 10-18). After extracting the core-vectors, P-CAV then constructs clusters (lines 19-35). As we can see from Algorithm 2, the core-vector detection and the cluster construction are performed by the thread-parallel manner.

4.2. *Thread-parallel Node Pruning*

As we described in Section 4.1, P-CAV first performs core-vector detection in a thread-parallel manner. Actually, the core-vector detection step is the most time-consuming one since the naïve algorithm CAV needs to compute all pairs of node-vectors in V. Thus, to speed up the core-vector detection step, we propose a thread-parallel node pruning technique that dynamically reduces unnecessary computations of pairs of node-vectors.

For efficiently detecting the unnecessary and redundant computations, P-CAV maintains an integer value, named *similar-degree* (sd), for each node-vector taken from the node-vector extraction. Formally, sd is defined as follows:

Definition 7 (Similar-Degree) *The similar-degree of node u, denoted $sd[u]$, is the number of node-vectors in V that have been determined to be included in ω-neighborhood of node u, i.e., $v \in N_u^\omega$ for $v \in V$ and $u \neq v$.*

Definition 7 indicates that node u should be a core-vector once $sd[u]$ reaches μ since if $sd[u] \geq \mu$ then we clearly hold $N_u^\omega \geq \mu$. That is, we do not need to compute cohesiveness of a node u for detecting a core-vector once the node has a similar-degree value that exceeds the user-specified parameter μ.

Based on Definition 7, P-CAV detects core-vectors from the node-vectors by a thread-parallel algorithm with node pruning. The pseudocode of the thread-parallel node pruning is shown in Algorithm (lines 10-18). Algorithm 2 (lines 10-18) detects all core-vectors included in V by using the node-pruning technique in a thread-parallel manner. As we can see from

Algorithm 2 Parallel Algorithm: P-CAV

Input: A graph $G = (V, E)$, and parameters, ω and μ;
Output: A set of clusters $\mathbb{C}$ and a set of outliers $\mathbb{O}$;

▷ **(1) Extracting node-vectors**
Obtain $\mathbf{B}$ by Equation (4);
for each $i \in V$ **do**
  for each $j \in V$ **do**
    Get $B_{ij} = \sum_{l=1}^{n} \lambda_l U_{il} U_{jl}$ by eigenvalue decomposition;
Get $p = |\{\lambda_1, \lambda_2, \ldots, \lambda_n \mid \lambda_i > 0, 1 \leq i \leq n\}|$;
for each $i \in V$ **do**
  for $l = 1$ to p **do**
    $r_i[l] = \sqrt{\lambda_l} U_{il}$ from Definition 2;
  Obtain $\mathbf{r}_i = (r_i[1], r_i[2], \ldots, r_i[p])^T$;

▷ **(2) Thread-parallel Node Pruning**
$\mathbb{C} = \emptyset$, $\mathbb{O} = \emptyset$;
for each node-vector $\mathbf{r}_i$ **do in thread-parallel**
  for each node-vector $\mathbf{r}_j$ **do**
    Get $sd[i]$ by Definition 7;
    if $sd[i] < \mu$ **then**
      Compute a cohesiveness between $\mathbf{r}_i$ and $\mathbf{r}_j$;
    else
      Regard $\mathbf{r}_i$ as a core-vector by Definition 4;
      break;

▷ **(3) Thread-parallel Cluster Construction**
Assign each core-vector $\mathbf{r}_i$ as a singleton partition P_i;
for each partition P_i **do in thread-parallel**
  for each partition P_j **do**
    if $\mathsf{find}(i) \neq \mathsf{find}(j)$ **then**
      if $\mathbf{r}_j \notin N_i^{\omega}$ **then** Compute a cohesiveness between $\mathbf{r}_i$ and $\mathbf{r}_j$;
      if $\mathbf{r}_j \in N_i^{\omega}$ **then** $P_i = P_i \cup P_j$ by $\mathsf{union}(i, j)$;
for each non-core-vector $\mathbf{r}_i$ **do in thread-parallel**
  for each core-vector $\mathbf{r}_j$ **do**
    if $\mathbf{r}_i \notin N_j^{\omega}$ **then** Compute a cohesiveness between $\mathbf{r}_i$ and $\mathbf{r}_j$;
    if $\mathbf{r}_i \in N_j^{\omega}$ **then** $P_i = P_i \cup \{i\}$;
for each P_i **do**
  if $|P_i| \geq 2$ **then**
    $\mathbb{C} = \mathbb{C} \cup \{P_i\}$;
for each non-partitioned node-vector $\mathbf{r}_i$ **do**
  $P_i = \{i\}$;
  $\mathbb{O} = \mathbb{O} \cup \{P_i\}$;
return $\mathbb{C}$ and $\mathbb{O}$;

Algorithm 2 (line 11), P-CAV first assigns each node-vector to each thread. In the thread, P-CAV computes pairs of node-vectors only if the assigned node-vector such that it has not determined as a core-vector. By following Definition 7, if $sd[u] \geq \mu$ then the node-vector of node u satisfies the core-vector condition shown in Definition 4. Meanwhile, if $sd[u] < \mu$, P-CAV needs to compute pairs of node-vectors to determine whether the node-vector of node u is core-vector or not. Hence, once P-CAV determines a node-vector is core-vector, it stops to compute the cohesiveness between the node-vector and the other ones (lines 17-18 in Algorithm 2).

4.3. *Thread-parallel Cluster Construction*

P-CAV finally constructs clusters over multiple threads. In order to maintain clusters among multiple threads, P-CAV employs *union-find tree* [25]. The union-find tree efficiently maintains a set of nodes-vectors partitioned into disjoint clusters by using two fundamental operators, called $\mathsf{find}(u)$ and $\mathsf{union}(u, v)$. $\mathsf{find}(u)$ looks up a cluster what does a node-vector of node u belong, and $\mathsf{union}(u, v)$ merges two node-vectors of node u and v into the same cluster. We can efficiently look up and merge clusters since both operators run at most $\Omega(A(n))$ times, where A is Ackermann function.

Algorithm 2 (lines 19-35) shows the pseudocode of the thread-parallel cluster construction. P-CAV first constructs clusters composed of core-vectors extracted by the previous step (lines 19-24 in Algorithm 2). After that, P-CAV attaches node-vectors, which are not core-vectors, to the clusters (lines 25-28). To avoid conflicts of union operators among multiple threads, P-CAV utilizes the atomic operation compare-and-swap (CAS) [26] before merging two clusters. Finally, P-CAV extracts a set of clusters $\mathbb{C}$ (lines 29-31), and it classifies the reminder node-vectors as outliers (lines 32-34).

5. Experimental Analysis

We conducted extensive experiments to evaluate the effectiveness of our proposed approaches, CAV and P-CAV. We designed our experiments to demonstrate that:

- **High Accuracy:** CAV and P-CAV outperform the state-of-the-art algorithms SFO [20] and ZN [21] in terms of clustering accuracy.
- **Efficiency:** P-CAV is more efficient than our naïve algorithm CAV on large-scale graphs.
- **Scalability:** P-CAV is scalable to the number of threads and edges.

5.1. *Experimental setup*

We compared the effectiveness of four algorithms including our proposed methods CAV and P-CAV.

- **CAV [22]:** Our proposed algorithm that we described in Section 3. Unless otherwise stated, we set $\omega = 20$ and $\mu = 2$.

- **P-CAV:** Our proposed algorithm that we described in Section 4. This algorithm is a thread-parallel extension of our previous approach CAV shown in Section 3. Unless otherwise stated, we set the number of threads as 14. Also, we set $\omega = 20$ and $\mu = 2$ as the default settings.
- **SFO:** The state-of-the-art graph clustering algorithm based on the modularity maximization [20]. This algorithm does not require any parameters, and it automatically determines the number of clusters.
- **ZN:** The state-of-the-art spectral algorithm for the modularity-based graph clustering [21]. Since this algorithm requires the number of clusters k as a user-specified parameter, we set k as the number of clusters included in the ground-truth of each dataset.

All experiments were conducted on a CentOS server with an Intel (R) Xeon (R) E5-2690 2.60 GHz CPU, which equips 14 physical cores, and 128 GB RAM.

5.1.1. *Datasets*

We evaluated the algorithms on both synthetic and real-world graphs detailed below.

Synthetic graphs: We evaluated the algorithms on synthetic graphs with their ground-truth clusters generated by LFR benchmark [27]. We first generated three graphs composed of 2,000 nodes, **LFR (0.1), LFR (0.5), and LFR (0.9)**, by varying a mixing parameter *mu* as 0.1, 0,5, and 0.9, respectively. The mixing parameter *mu* controls a fraction of neighbor nodes included in other clusters for each node; if the value of *mu* decreases, the clusters are explicitly separated in the graph. The other parameters of LFR benchmark, average degree, maximum degree, and clustering coefficient, were fixed at 20, 50 and 0.4, respectively.

In addition to the above graphs, we generated synthetic graphs with the different graph sizes for the scalability test. We varied the number of nodes from 1,000 to 16,000, where the mixing parameter, the average degree, the maximum degree, and the clustering coefficient, were fixed at 0.1, 20, 50 and 0.4, respectively.

Real-world graphs: We also evaluated the algorithms on two real-world graphs that have ground-truth clustering results. The details of the real-world graphs are listed as follows:

- **Email [28]:** This is a graph generated by e-mail communication data from a large European research institution. The number of nodes and edges are 1,005 and 25,571, respectively.
- **HPEC [29]:** This graph is published by Graph Challange competition collocated with the IEEE High Performance Extreme Computing Conference 2017. The graph has 20,000 nodes and 408,778 edges.

5.2. *Clustering Accuracy*

We evaluated the accuracy of the clustering results produced by the algorithms by using an information-theoric metric, *normalized mutual information (NMI)* [30]. NMI compares two

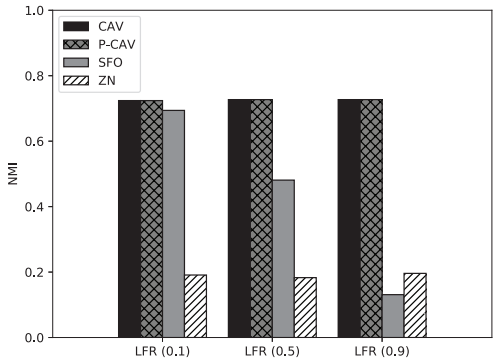

Fig. 2. NMI on synthetic graphs.

clustering results, and it returns 1 if two clustering results are completely the same, otherwise 0. We compared the clustering results included in the ground-truth of each dataset with the results produced by each algorithm that we performed in the experiments.

5.2.1. *Evaluation on synthetic graphs*

Figure 2 shows NMI values of the algorithms for each synthetic graph. As we can see from the results, our proposal, CAV, shows much higher NMI values than the other algorithms under all conditions examined. Of particular interest, CAV achieved the most significant improvement on LFR (0.9); it shows 60% and 53.3% higher accuracy than SFO and ZN, respectively. Since LFR (0.9) is generated by a large *mu* value, LFR (0.9) has many edges among clusters included in the graph. However, as we described in Section 2, the state-of-the-art modularity-based algorithms like SFO and ZN fails to extract such densely connected clusters. This is because these methods explore a set of nodes based on the modularity maximization that does not handle the cohesiveness of each cluster. Thus, SFO and ZN fail to extract fine-grained clustering results from the graphs. In contrast to SFO and ZN, our proposed algorithm is designed to capture the cohesiveness of each cluster by using the cohesiveness-aware vector partitioning.

Our experiments also considered the effects of user-specified parameters, ω and μ, on CAV. To evaluate the effects of the parameters, we fixed one of the two parameters and varied the other one on the synthetic datasets. Figure 3 shows NMI values by varying ω with fixed parameter, $\mu = 2$. As shown in Figure 3, CAV gradually decreases NMI values as ω increases. This is because it is difficult for a large ω setting to exclude sparsely connected nodes from the clusters. Hence, as demonstrated in Figure 3, smaller ω values are effective for improving the clustering accuracy. Similarly, in Figure 4, we demonstrate the effects of μ by varying μ values as 2, 10, 50, and 100 with fixed ω value, *i.e.,* $\omega = 20$. As we can see from Figure 4, NMI values are almost stable under all μ settings that we examined.

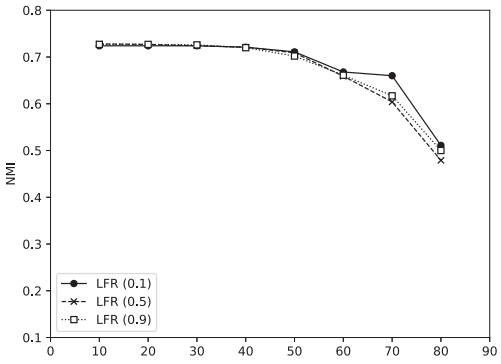

Fig. 3. Effect of ω on synthetic graphs.

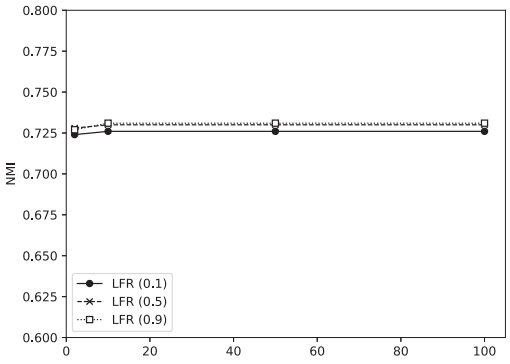

Fig. 4. Effect of μ on synthetic graphs.

5.2.2. *Evaluation on real-world graphs*

We evaluated the clustering accuracy on the real-world graphs. Figure 5 shows NMI values of the algorithms on Email and HPEC datasets. As well as the results in Section 5.2.1, our proposed method achieved higher accuracy than the state-of-the-art methods, SFO and ZN. Specifically, CAV showed 19.5% and 55.8% higher NMI values than those of SFO and ZN, respectively. From these results, we experimentally verified the effectiveness of our approach on real-world datasets.

Similar to the results on the synthetic graphs, we also tested the effects of the user-specified parameters ω and μ on real-world graphs. In Figure 6, we varied ω from 10 to 80 with $\mu = 2$. Also, Figure 7 shows the results when we set μ as 2, 10, 50, and 100 with $\omega = 20$. As we can see from the results, the small ω settings show better accuracy than larger ones, and the results varying μ are almost stable. Therefore, ω and μ should be the small values for improving the clustering accuracy on real-world datasets.

5.3. *Efficiency*

We here evaluate the efficiency of P-CAV compared with CAV. Figure 8 and 9 show the running time of each algorithms, respectively. As shown in Figure 8 and 9, P-CAV successfully reduces the computation time of CAV from 87.1% to 96.4%. In the best case, P-CAV runs 27.93 times faster than CAV under the experimental settings that we examined. As we described in Section 4, P-CAV employs thread-parallel node pruning technique in order to dynamically removes unnecessary computations in the cohesiveness-aware vector partitioning. In contrast, as shown in Algorithm 1, our naïve algorithm CAV requires $O(n^3)$ times to complete the cohesiveness-aware vector partitioning since CAV needs to perform exhaustive explorations for all node-vectors extracted from a given graph. Thus, our extended approach P-CAV achieves faster clustering than CAV.

5.4. *Scalability*

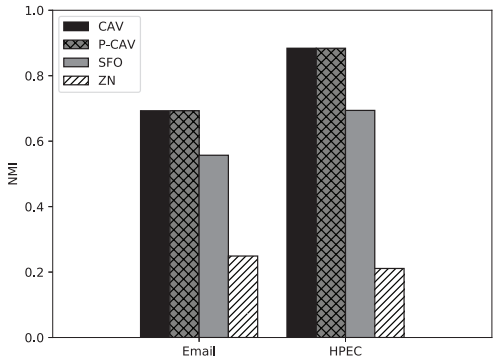

Fig. 5. NMI on real graphs.

We finally assessed scalability test of our proposed approach P-CAV in Figure 10 and 11 by increasing the number of nodes and threads, respectively. In Figure 11, we used five synthetic graphs with different node sizes by LFR-benchmark; we generated graphs composed of 1,000, 2,000, 4,000, 8,000, and 16,000 nodes with fixed the other parameters as we described in Section 5.1.1. Meanwhile, in Figure 10, we generated used LFR (0.1) dataset that we used in Figure 11. As we can see from the results, the runtimes of P-CAV has near-linear scalability in terms of the number of nodes and edges even though the naïve algorithm requires $O(n^3)$ times for the clustering. These results verify that P-CAV is scalable for large-scale graphs.

6. Related work

The problem of finding densely connected groups from graph-structured data has been studied in a few decades [31]. Modularity, proposed by Newman and Girvan [10], is a widely used graph clustering metric that evaluates a clustering result from a global perspective. The heart of modularity is to find groups of nodes that have a lot of inner-group edges and few inter-group edges. Thus, the modularity-based algorithms are designed to find groups of nodes that maximize modularity Q shown in Definition 1. However, the optimization of modularity Q is known as NP-hard problem [32]. This had led to the introduction of approximation approaches. Here, we review some of the more successful approximation methods in Web-based applications and systems.

6.1. *Greedy optimization algorithms*

In order to avoid the expensive cost of the modularity-based algorithms, the greedy optimization algorithms have been proposed in recent decades. The main idea underlying this class of algorithms is to evaluate a gain of the modularity Q that we obtain after merging two clusters. Newman method [32] is one of the representative algorithms in greedy optimization methods. Newman method explores the best increase of the gain among all pairs of clusters, and then it greedily merges two clusters with the largest gain into the same cluster. As a result,

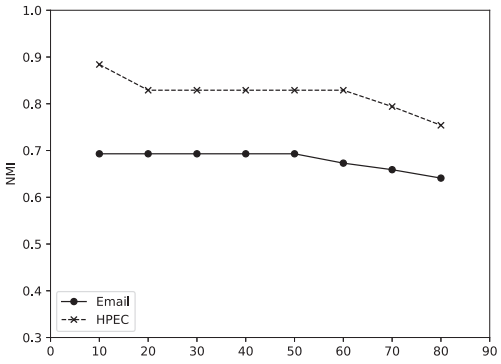

Fig. 6. Effect of ω on real graphs.

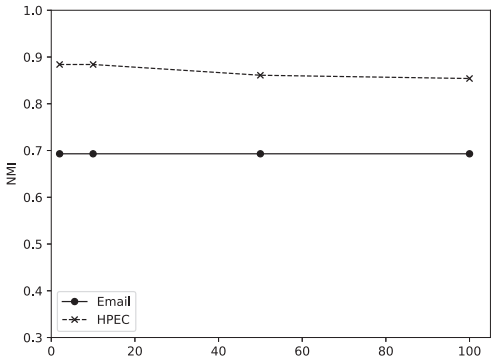

Fig. 7. Effect of μ on real graphs.

it produces reasonable clusters with hierarchical structures, which represent the history of merges. Despite the effectiveness in avoiding the NP-complete problem, it still requires high computing cost. To reduce the computational cost of Newman method, several algorithms including CNM [33], Louvain [23], and the state-of-the-art method SFO [20], have been proposed. CNM [33] is an index-based approach method, and on the other hand, Louvain [23], and SFO [20] employ node-aggregation approaches for the efficient maximization of the modularity Q. These methods successfully reduce the running time of the greedy computation besides maximizing the modularity Q.

Although the greedy optimization algorithms are useful for maximizing the modularity Q, they still suffer from the resolution limit problems that we described in Section 2 since they do not maintain the cohesiveness of each cluster. Different from the algorithms, our proposed methods employ the cohesiveness-aware vector partitioning technique in order to exclude sparsely connected nodes from the clusters. As a result, as shown in Section 5.3, our proposed methods achieve higher accuracy than the state-of-the-art greedy method SFO on both synthetic and real-world datasets.

6.2. *Spectral partitioning algorithms*

Another class of the modularity-based clustering is the spectral partitioning algorithms. Similar to our proposed method, the spectral partitioning algorithms first extract a vectorized representation for each node via spectral analysis of the given graph. After that, they partition the vectors into k disjoint groups so that the groups maximize the modularity Q by using an arbitrary vector partitioning algorithm. Richardson *et al.* [34] proposed a divide-and-conquer method that divides the space of vectors into two subspace so that smaller edges split. Their method continues to divide the space of vectors until further subdivision gives no improvement of modularity Q. Zhang and Newman [21] also proposed another type of vector partitioning algorithm inspired by k-means method [35]. Their proposed method first conducted a distance measure as the gain of modularity that is obtained after merging two vectors in the

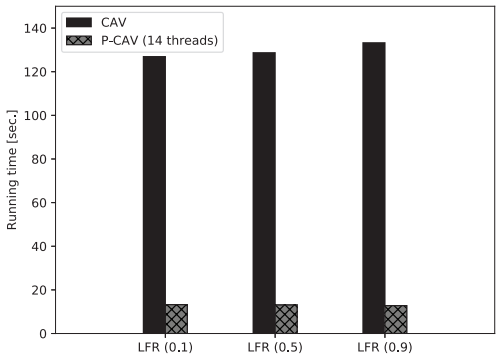

Fig. 8. Running time on synthetic graphs.

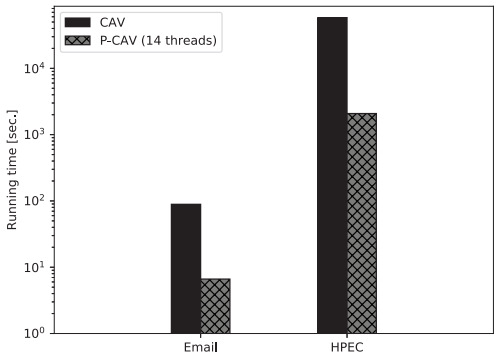

Fig. 9. Running time on real-world graphs.

same cluster. Then, by using the distance, it performs k-means like algorithm and divides the vectors into k disjoint partitions so that the partitions maximize the modularity Q.

Since these methods are designed as the modularity maximization problem via spectral analysis, they also fail into the resolution limit problem [14]. Furthermore, these methods require the number of clusters k as a user-specified parameter even though we do not typically know the number of clusters before the clustering. Different from the above spectral partitioning algorithms, our proposed method can avoid to group sparsely connected nodes from clusters by the cohesiveness-aware approach. Besides, it does not require to set the number of clusters k for the clustering. Hence, as shown in Section 5.3, our proposed methods show higher accuracy than the state-of-the-art spectral partitioning method.

7. Conclusion

This paper addressed the resolution limit problem of the modularity-based graph clustering. In order to overcome the resolution limit problem, we focused on the cohesiveness of each cluster, and we proposed novel algorithms, named CAV and P-CAV. Our proposed methods, CAV and P-CAV, are consisted of two major phases: (1) it represents each node as a p-dimensional vector by applying the spectral analysis to the modularity Q, (2) it performs the vector partitioning algorithm by capturing the cohesiveness of each partition. Also, P-CAV employs thread-based parallelization for further improving the clustering speed of CAV in the above vector partitioning phase. Our extensive evaluation using several datasets demonstrated that CAV and P-CAV achieve higher accuracy than the state-of-the-art algorithms. Furthermore, we experimentally verified that P-CAV successfully reduces the running time of CAV for large-scale graphs. The modularity-based graph clustering algorithm is now an essential tool for various Web-based applications and systems even though it suffers from the resolution limit of clustering results. By providing the cohesiveness-aware approaches that can capture fine-grained clusters from a given graph, the proposal will help to improve the effectiveness of current and future applications.

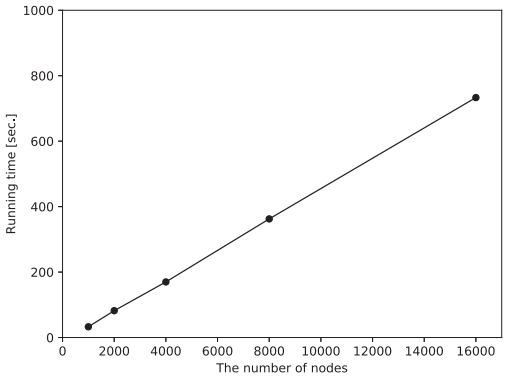

Fig. 10. Scalability by varying the number of nodes.

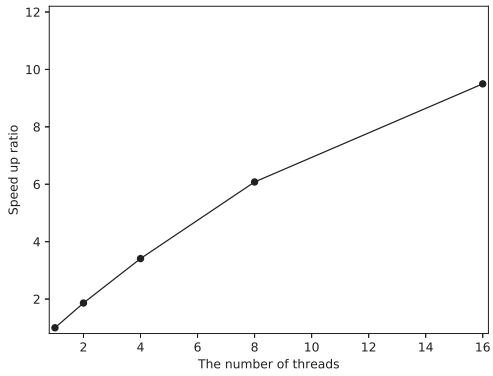

Fig. 11. Scalability by varying the number of threads.

Acknowledgements

This work was partially supported by JST ACT-I, JSPS KEKENHI Early-Carrer Scientists Grant Number JP18K18057, and Interdisciplinary Computational Science Program in CCS, University of Tsukuba.

References

1. T. Sato, H. Shiokawa, Y. Yamaguchi, and H. Kitagawa, "FORank: Fast ObjectRank for Large Heterogeneous Graphs," in *Companion Proceedings of the The Web Conference 2018*, pp. 103–104, 2018.
2. H. Shiokawa, Y. Fujiwara, and M. Onizuka, "SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs," *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, vol. 8, pp. 1178–1189, July 2015.
3. Y. Fujiwara, M. Nakatsuji, H. Shiokawa, Y. Ida, and M. Toyoda, "Adaptive message update for fast affinity propagation," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2015, pp. 309–318, 2015.
4. T. Takahashi, H. Shiokawa, and H. Kitagawa, "SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors," in *Proceedings of the 2nd ACM SIGMOD Workshop on Network Data Analytics (NDA 2017)*, pp. 6:1–6:7, 2017.
5. Z. Li, S. Zhang, R.-S. Wang, X.-S. Zhang, and L. Chen, "Quantative Function for Community Detection," *Physical Review*, vol. E 77, no. 036109, 2008.
6. T. C. Zhou, H. Ma, M. R. Lyu, and I. King, "UserRec: A User Recommendation Framework in Social Tagging Systems," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010.
7. J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888–905, August 2000.
8. 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, December 1998.
9. Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura, "Fast Lasso Algorithm via Selective Coordinate Descent," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 1561–1567, 2016.

10. M. E. J. Newman and M. Girvan, "Finding and Evaluating Community Structure in Networks," *Physical Review*, vol. E 69, no. 026113, 2004.
11. J. Weng and B. Lee, "Event Detection in Twitter," in *Proceedings of the 5th International Conference on Weblogs and Social Media (ICWSM)*, 2011.
12. C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to Route with Sparse Trajectory Sets," in *Proceedings of the 34th IEEE Conference on Data Engineering*, pp. 1073–1084, 2018.
13. J. O. Garcia, A. Ashourvan, S. Muldoon, J. M. Vettel, and D. S. Bassett, "Applications of Community Detection Techniques to Brain Graphs: Algorithmic Considerations and Implications for Neural Function," *Proceedings of the IEEE*, vol. 106, pp. 846–867, May 2018.
14. S. Fortunato and M. Barthelemy, "Resolution Limit in Community Detection," *Proceedings of the National Academy of Sciences*, Jan 2007.
15. X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "SCAN: A Structural Clustering Algorithm for Networks," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, (New York, NY, USA), pp. 824–833, ACM, 2007.
16. A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One Trillion Edges: Graph Processing at Facebook-scale," *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, vol. 8, pp. 1804–1815, August 2015.
17. H. Shiokawa, T. Takahashi, and H. Kitagawa, "ScaleSCAN: Scalable Density-based Graph Clustering," in *Proceedings of the 29th International Conference on Database and Expert Systems Applications*, DEXA, pp. 18–34, 2018.
18. S. Muff, F. Rao, and A. Caflisch, "Local Modularity Measure for Network Clusterizations," *Physical Review*, vol. E 72, no. 056107, 2005.
19. A. Costa, "Comment on "Quantitative Function for Community Detection"," *CoRR*, vol. abs/1409.4063, 2014.
20. H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast Algorithm for Modularity-based Graph Clustering," in *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1170–1176, June 2013.
21. X. Zhang and M. E. J. Newman, "Multiway Spectral Community Detection in Networks," *Physical Review E*, vol. 92, p. 052808, Nov 2015.
22. H. Shiokawa and Y. Futamura, "Graph Clustering via Cohesiveness-aware Vector Partitioning," in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, iiWAS2018, (New York, NY, USA), pp. 33–40, ACM, 2018.
23. V. Blondel, J. Guillaume, R. Lambiotte, and E. Mech, "Fast Unfolding of Communities in Large Networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.
24. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and Identifying Communities in Networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. 9, pp. 2658–2663, 2004.
25. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.
26. M. Herlihy, "Wait-free Synchronization," *ACM Transaction on Programming Languages and Systems*, vol. 13, pp. 124–149, Jan. 1991.
27. A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the Overlapping and Hierarchical Community Structure in Complex Networks," *New Journal of Physics*, vol. 11, no. 3, p. 033015, 2009.
28. J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph Evolution: Densification and Shrinking Diameters," *ACM Transactions on Knowledge Discovery from Data (ACM TKDD)*, vol. 1, March 2007.
29. E. K. Kao, V. Gadepally, M. B. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, D. Staheli, and S. Smith, "Streaming Graph Challenge: Stochastic Block Partition," in *2017 IEEE High Performance Extreme Computing Conference, (HPEC 2017)*, pp. 1–12, 2017.
30. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York,

NY, USA: Cambridge University Press, 2008.
31. M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph partitioning for distributed graph processing," *Data Science and Engineering*, vol. 2, pp. 94–105, Mar 2017.
32. M. E. J. Newman, "Fast Algorithm for Detecting Community Structure in Networks," *Physical Review*, vol. E 69, no. 066133, 2004.
33. A. Clauset, M. E. J. Newman, , and C. Moore, "Finding Community Structure in Very Large Networks," *Physical Review*, vol. E 70, no. 066111, 2004.
34. T. Richardson, P. J. Mucha, and M. A. Porter, "Spectral Tripartitioning of Networks," *Physical Review E*, vol. 80, p. 036111, Sep 2009.
35. J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, University of California Press, 1967.