

Regular Paper

Fast and Parallel RankClus Algorithm based on Dynamic Rank Score Tracking

KOTARO YAMAZAKI^{1,†1,a)} SHOHEI MATSUGU^{1,b)} HIROAKI SHIOKAWA^{2,c)} HIROYUKI KITAGAWA^{2,d)}

Received: December 10, 2019, Accepted: February 14, 2020

Abstract: Given a bi-type information network, which is an extended model of well-known bipartite graphs, how can clusters be efficiently found in graphs? Graph clustering is now a fundamental tool to understand overviews from graph-structured data. The RankClus framework accurately performs clustering for bi-type information networks using ranking-based graph clustering techniques. It integrates graph ranking algorithms such as PageRank or HITS into graph clustering procedures to improve the clustering quality. However, this integration incurs a high computational cost to handle large bi-type information networks since RankClus repeatedly computes the ranking algorithm for all nodes and edges until the clustering procedure converges. To overcome this runtime limitation, we present a novel RankClus algorithm that reduces the running time for large bi-type information networks. Our proposed method employs dynamic graph processing techniques into the iterative ranking procedures included in RankClus. By dynamically updating ranking results, our proposal reduces the number of computed nodes and edges during repeated ranking procedures. For further improving the efficiency, we also present a parallel implementation of our proposed algorithm by using thread-based parallelization on a modern manycore CPU. We experimentally verify using real-world datasets that our fast and parallel proposed methods successfully reduces the running time while maintaining the clustering quality of RankClus.

Keywords: RankClus, Graph Mining, Clustering

1. Introduction

Graphs can handle schema-less and complex real-world phenomena. They represent data entities and their relationships using nodes and edges, respectively. Due to recent advances in information sciences and web technologies, large-scale graphs are ubiquitous in diverse application domains from the Internet to biological networks [19]. As graph sizes are constantly increasing, it is apparent that techniques to analyze large graphs are needed. To understand large-scale graphs, graph cluster analysis (community detection) is an important data mining tool in various research areas such as web engineering, social analysis, and bioinformatics. A cluster can be regarded as a group of nodes that are densely connected with each other but are sparsely connected between different groups. By discovering the hidden cluster structures included in real-world graphs, not only can raw data be overviewed but also the interrelationships among nodes can be discovered. Consequently, identifying clusters included in a graph has become an interesting and important problem.

The problem of finding clusters in a graph has been studied

for several decades in many areas, especially in physics and computer science. Traditional clustering algorithms such as graph partitioning [18], modularity clustering [19], [20], and density-based methods [21], [23], [27] are natural choices for this problem. Basically, these algorithms are designed to compute homogeneous graphs. However, real-world graphs are generally more complex and each node can have an attribute on various real-world applications. Thus, recent applications model such graphs as bi-type information networks [26], which are special class of heterogeneous graphs. Unfortunately, traditional algorithms can not handle bi-type information networks since they ignore attributes attached to each node, and explore only densely connected subgraphs. Thus, graph clustering on bi-type information networks remains a challenging task.

To achieve graph clustering on bi-type information networks, Sun et al. presented the RankClus framework [26]. RankClus initially divides node attributes on bi-type information networks into two groups: target type and attribute type. Then, it clusters target-type nodes using attribute-type nodes as support information. Specifically, RankClus integrates a clustering procedure with a node ranking technique such as PageRank [4] or HITS [10] to characterize target-type nodes by attribute-type nodes. The framework performs clustering and ranking consecutively. It gradually improves the clustering quality and ranking quality in an iterative manner. By using attribute-type nodes as support information of target-type nodes, RankClus successfully uncovers

¹ Graduate School of System and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan

² Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan

^{†1} Presently with ZOZO Technologies, Inc., Shibuya, Tokyo 150–0001, Japan

^{a)} k.yamazaki@kde.cs.tsukuba.ac.jp

^{b)} matsugu@kde.cs.tsukuba.ac.jp

^{c)} shiokawa@cs.tsukuba.ac.jp

^{d)} kitagawa@cs.tsukuba.ac.jp

This work was done when Kotaro Yamazaki was at University of Tsukuba.

highly accurate clusters included in the target-type nodes [26].

Although the RankClus framework can effectively uncover clusters on various bi-type information networks such as bibliographic networks, web graphs, and biological networks, the framework requires expensive computational costs to handle large-scale bi-type information networks because the RankClus framework iteratively performs clustering and ranking procedures until stable clusters are found. Specifically, the framework starts its computation from randomly partitioned clusters, and then obtains a rank score for each node in a graph by a node ranking algorithms such as PageRank or HITS. Next, the framework clusters target-type nodes again based on the assumption that a node appropriate for the cluster has a higher rank score. Otherwise, it has a lower rank score. Following the above assumption, RankClus performs traditional clustering [11] by the rank score of each node in the clusters. Finally, RankClus framework performs ranking and the clustering procedures iteratively until the clusters converge. The RankClus framework must compute the rank scores of all nodes in each ranking procedure. Each procedure requires $O(n^2)$ times, where n is the number of nodes in a bi-type information network. Hence, RankClus requires exhaustive computational costs to find converged clusters in a large-scale bi-type information networks. Recently, Yamazaki et al., proposed edge-pruning approaches [29], [30] to reduce the computational costs of the RankClus framework. However, the improvements of efficiency are limited and these approaches sacrifice clustering quality compared to the RankClus framework.

1.1 Our Approaches and Contributions

We focus on improving the clustering efficiency of the RankClus framework for a large-scale bi-type information network. To tackle this problem, we present a novel algorithm for RankClus that effectively reduces the exhaustive computational costs without degrading the clustering quality. Our proposed algorithm introduces a rank score tracking method into the ranking procedure of the RankClus framework. Instead of exhaustively computing all nodes and edges, the rank score tracking method dynamically updates rank scores of only nodes whose rank score significantly change. That is, our proposed algorithm reduces the number of computed nodes and edges during the ranking procedure by integrating a rank score tracking method into the RankClus framework. In addition to the dynamic rank score tracking, we discuss an extended the dynamic rank score tracking algorithm for further improving the efficiency of our proposed algorithm. We employ thread-based parallelization techniques to reduce the computational time of our dynamic rank score tracking method since our algorithm consists of several independent loop-blocks. By applying loop-level parallelization, the our parallel algorithm significantly reduces the computational time while keeping the accuracy of clustering. As a result, our proposed methods have the following characteristics:

- **Efficient:** Our proposals outperform the RankClus [26] and the state-of-the-art algorithm [29] in terms of running time for real-world bi-type information networks.
- **Effectiveness:** For further improving for the clustering speed, we extended our algorithm to utilize thread-based par-

allelization on a modern manycore processor.

- **Highly accurate:** Although our proposed algorithms do not compute the entire graph, their clustering results are more accurate than those of the state-of-the-art algorithm [29]. Additionally, our proposals output are almost the same as the clustering results of the RankClus for real-world graphs.
- **Easy to deploy:** Our proposed methods require fewer user-specified parameters than the state-of-the-art algorithm [29]. This property allows users to deploy the graph clustering algorithm in real-world applications easily.

Our experimental analysis shows that our proposed algorithms are 3.5 times faster than the competitive algorithms [26], [29], [30], while outputting almost the same clustering results as RankClus. The RankClus framework can enhance the qualities of various web-based applications, however it is difficult to apply to large-scale bipartite graphs due to its exhaustive computations. Our efficient algorithms will improve the effectiveness of the RankClus framework in a broader range of applications.

The rest of this paper is organized as follows. Section 2 briefly discusses the background. Section 3 presents our proposed algorithm, and Section 4 discusses our experimental analysis using real-world datasets. Section 5 we overviews related works. Finally, Section 6 concludes this paper.

2. Preliminary

This section formally defines the notations and briefly introduces the RankClus framework.

2.1 Bi-type Information Network

We first define a *bi-type information network*, which is an input data model of RankClus and our proposed algorithm.

Definition 1 (Bi-type Information Network) Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$ be a bi-type information network, \mathbb{V} and \mathbb{E} are defined as $\mathbb{V} = X \cup Y$ and $\mathbb{E} = \{\langle o_i, o_j \rangle | o_i, o_j \in X \cup Y\}$, respectively. Note that $X = \{x_1, x_2, \dots, x_m\}$ and $Y = \{y_1, y_2, \dots, y_n\}$ are two sets of nodes such that $X \cap Y = \emptyset$, where $m = |X|$ and $n = |Y|$. A bi-type information network is defined as a special class of a heterogeneous graphs. Based on Definition 1, the RankClus framework assigns two types to X and Y : *target type* and *attribute type*. RankClus framework clusters only the target-type nodes using attribute-type nodes as a clustering guide.

Figure 1 is a conference–author network modeled as a bi-type information network in Definition 1. The graph has two types of nodes X and Y that represent sets of conferences and authors, respectively. If author y_j has published at least one paper at conference x_i , nodes x_i and y_j are linked. Also, two authors y_i and y_j are linked only if y_i is a co-author of y_j , and vice versa.

Here, we define the weight of an edge between nodes o_i and o_j as w_{o_i, o_j} , and we represent an adjacency matrix of a bi-type information network is presented as $W_{(m+n) \times (m+n)} = \{w_{o_i, o_j}\}$. For simplicity, we decompose $W_{(m+n) \times (m+n)} = \{w_{o_i, o_j}\}$ into four blocks: W_{XX} , W_{XY} , W_{YX} and W_{YY} . Each block is a subgraph of the given network among subscript types. That is, W is defined as:

$$W = \begin{pmatrix} W_{XX} & W_{XY} \\ W_{YX} & W_{YY} \end{pmatrix}.$$

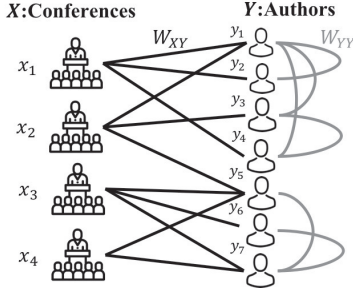


Fig. 1 Example of a bi-type information network: This graph has two types of nodes X and Y , which represent sets of conferences and authors, respectively. If author y_j has published at least one paper at conference x_i , there is a link between node x_i and y_j . Also, two authors y_i and y_j are linked only if y_i is a co-author of y_j , and vice versa.

In Fig. 1, matrixes W_{XY} and W_{YY} are defined as:

$$W_{XY}(i, j) = p_{ij}, \text{ for } i = 1, 2, \dots, m; \text{ and } j = 1, 2, \dots, n,$$

where p_{ij} is the number of papers published by author y_j at conference x_i .

$$W_{YY}(i, j) = a_{ij}, \text{ for } i = 1, 2, \dots, m; \text{ and } j = 1, 2, \dots, n,$$

where a_{ij} is a number of papers written by authors y_i and y_j . Clearly, $W_{YX} = W_{XY}^T$ and $W_{XX} = O$.

2.2 RankClus Framework

The RankClus framework [26] finds clusters from the target nodes using the attribute nodes as a guide for the clustering procedure. To characterize target-type nodes, RankClus performs ranking procedures for all nodes included in a given bi-type information network before extracting clusters. By using rank scores obtained by the ranking procedure, RankClus then clusters target-type nodes. Once a clustering result is obtained, RankClus performs the ranking procedure again. It iteratively continues the above processes until stable clusters are obtained.

The pseudo-code of the RankClus framework is shown in Algorithm 1, which takes bi-type information network \mathbb{G} and a number of cluster K as inputs. The RankClus performs the following workflow until its clustering results converges:

- (1) **Initialization:** Randomly partition target-type nodes into K clusters and construct K subgraphs.
- (2) **Ranking procedure:** Computes rank scores of all nodes in each subgraph.
- (3) **Clustering procedure (i):** Constructs a K -dimensional vector for each node in target-type from the rank scores computed in (2).
- (4) **Clustering procedure (ii):** Cluster a set of K -dimensional vectors obtained by (3).
- (5) Repeat (2) to (4) until the clustering results converge.

2.2.1 Ranking procedure

RankClus divides target-type nodes X into K clusters. Then it then constructs K subgraphs. Let X_i be a subset of X all of whose nodes are included in i -th cluster ($1 \leq i \leq K$). Subgraph \mathbb{G}_i is defined as $\mathbb{G}_i = \langle \mathbb{V}_i, \mathbb{E}_i \rangle$, where $\mathbb{V}_i = \{X_i \cup Y\}$ and $\mathbb{E}_i = \{(o_i, o_j) | o_i, o_j \in X_i \cup Y\}$. That is, $\mathbb{G} = \bigcup_i^K \mathbb{G}_i$. **Figure 2** shows an example of a subgraph where X is partitioned into $\{x_1, x_2\}$ and $\{x_3, x_4\}$, and their corresponding subgraphs are built.

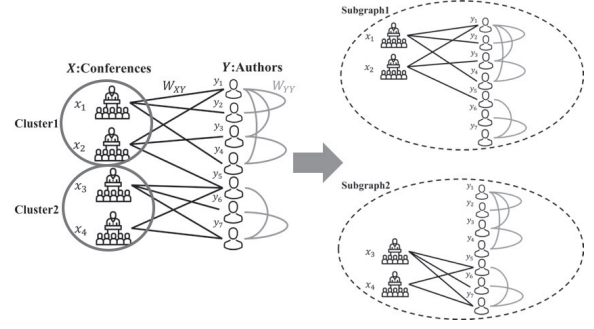


Fig. 2 Example of subgraph construction using Fig. 1: In this example, we set $K = 2$ and partition $X = \{x_1, x_2, x_3, x_4\}$ into two subsets $\{x_1, x_2\}$ and $\{x_3, x_4\}$. Then a subgraph is constructed for each subset.

Algorithm 1 RankClus

Input: $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, and K

Output: $X_i (i = 1, 2, \dots, K)$, \vec{r}_X , and \vec{r}_Y for each X_i

```

// Step 0: Initialization
1:  $t = 0$ .
2: Generate initial  $K$  clusters  $X_1^{(0)}, X_2^{(0)}, \dots, X_K^{(0)}$ .
3: repeat
    // Step 1: Ranking for each cluster
4: Construct subgraphs  $\mathbb{G}_i(t)$  from  $X_i(t)$ , and  $Y$ .
5: for  $i = 1$  to  $K$  do
6:   for each  $x_j \in X_i$  and  $y_s \in Y$  do
7:     Compute rank score  $\vec{r}_X(j, i)$  and  $\vec{r}_Y(s, i)$ .
8:   end for
9: end for
    // Step 2: Get new attribute
10: for  $i = 1$  to  $K$  do
11:   for each  $x_j \in X_i$  do
12:     Estimate  $\pi_{ji}$  by Definition 2.
13:   end for
14:   Determine a centroid vector  $\vec{s}_{X_i}$ .
15: end for
    // Step 3: Assign  $x_j$  to cluster
16: for each  $x_j \in X$  do
17:   for  $i = 1$  to  $K$  do
18:     Compute a distance  $D(j, k)$ .
19:   end for
20:   Obtain  $k_0 = \arg \min_k D(j, k)$ .
21:    $X_{k_0}^{(t+1)} = X_{k_0}^{(t)} \cup \{x_j\}$ .
22: end for
23:  $t = t + 1$ .
24: until No clusters are updated.
    
```

Let $\vec{r}_X(x, k)$ and $\vec{r}_Y(y, k)$ be the rank scores of $x \in X$ and $y \in Y$ in \mathbb{G}_k , respectively. For each subgraph, RankClus performs an arbitrary node ranking method. Hence, $\sum_{x \in X} \vec{r}_X(x, k) = 1$ and $\sum_{y \in Y} \vec{r}_Y(y, k) = 1$. Herein the node ranking method is the Personalized PageRank (PPR) algorithm [9]. Let $\vec{r}_{PPR}(i, k) \in \mathbb{R}^{(m+n)}$ be the rank score vector for $i \in \mathbb{V}$ in \mathbb{G}_k , $\vec{b} \in \mathbb{R}^{(m+n)}$ be a preference vector where each element is equal to $\frac{1}{m+n}$, and $P \in \mathbb{R}^{(m+n) \times (m+n)}$ be a transition matrix obtained by normalizing each column of W . PPR obtains a rank score vector $\vec{r}_{PPR}(i, k)$ by performing the following equation until $\vec{r}_{PPR}(i, k)$ converges.

$$\vec{r}_{PPR}(i, k) = \alpha P \vec{r}_{PPR} + (1 - \alpha) \vec{b},$$

where $\alpha \in [0, 1]$. Once $\vec{r}_{PPR}(i, k)$ is obtained for all subgraphs, the rank scores $\vec{r}_X(x, k)$ and $\vec{r}_Y(y, k)$ are computed as:

$$\vec{r}_X(x, k) = \frac{\vec{r}_{PPR}(x, k)}{\sum_{x \in X_k} \vec{r}_{PPR}(x, k)}, \vec{r}_Y(y, k) = \frac{\vec{r}_{PPR}(y, k)}{\sum_{y \in Y} \vec{r}_{PPR}(y, k)}$$

2.2.2 Clustering Procedure

RankClus updates K partitions of X in the clustering proce-

ture. First, it estimates the posterior probability $\pi_{i,k}$ that represents the probability of target-type node x_i belonging to cluster k using the rank scores obtained in the ranking procedure. The posterior probability is formally defined below.

Definition 2 (Posterior probability $\pi_{i,k}$) Let $\pi_{i,k}$ be the posterior probability of x_i belonging to cluster k , $\pi_{i,k}$ is defined as:

$$\pi_{i,k} = p(k|x_i) = \frac{\vec{r}_{(X|X_k)}(i, k)p(k)}{\sum_{l=1}^K \vec{r}_{(X|X_k)}(i, l)p(l)},$$

where $p(k)$ is the probability estimated by EM-algorithm, and

$$\vec{r}_{(X|X_k)}(x, k) = \frac{\sum_{j=1}^n W_{XY}(x, j)r_Y^*(j, k)}{\sum_{i=1}^m \sum_{j=1}^n W_{XY}(i, j)r_Y^*(j, k)}.$$

After that, the framework constructs K -dimensional feature vector $s_{x_i} = (\pi_{i,1}, \pi_{i,2}, \dots, \pi_{i,K})$ for each $x_i \in X$. Then, RankClus determines K cluster centroids, which are defined below:

Definition 3 (Cluster centroid) Let \vec{s}_{X_i} be a vector that represents a cluster centroid of cluster X_i ; \vec{s}_{X_i} is defined as:

$$\vec{s}_{X_i} = \frac{\sum_{x \in X_i} \vec{s}(x)}{|X_i|}.$$

Finally, the RankClus framework updates K clusters by assigning $x_i \in X$ into a cluster X_k that shows the smallest distance between s_{x_i} and s_{X_k} using the following definition.

Definition 4 (Distance) Let $D(i, k)$ be the distance between node x_i and cluster X_k . $D(i, k)$ is defined as:

$$D(i, k) = 1 - \frac{\sum_{l=1}^K \vec{s}_{x_i}(l)\vec{s}_{X_k}(l)}{\sqrt{\sum_{l=1}^K (\vec{s}_{x_i}(l))^2} \sqrt{\sum_{l=1}^K (\vec{s}_{X_k}(l))^2}}.$$

From Definition 2, estimating the posterior probability is not expensive since we can get $\vec{r}_{(X|X_k)}(i, k)$ and $p(k)$ in $O(1)$. Also, updating K clusters requires $O(mK)$ times, which is a smaller computational cost than the ranking part. Thus, the clustering part does not have a dominant cost in the RankClus framework.

3. Proposed Method

Our goal is to efficiently find clusters from target-type nodes without sacrificing the clustering quality compared to RankClus. Since the clustering procedure does not have a dominant cost, we attempt to reduce the computational cost of the ranking procedure, which entails exhaustive computations for all nodes and edges included in a bi-type information network.

3.1 Basic Ideas

RankClus computes rank scores for all nodes included in each subgraph \mathbb{G}_i . Then the framework updates $X_i \subset \mathbb{V}_i$ by following the clustering procedure. It should be noted that RankClus iterates the ranking and clustering procedures until the clustering results converge. Thus, each subgraph \mathbb{G}_i can be regarded as a graph that dynamically evolves X_i and E_i . The RankClus framework (1) inserts new nodes and edges into \mathbb{G}_i , and/or (2) removes several nodes and edges from \mathbb{G}_i in each iteration.

The main idea underlying our proposed methods are to handle the above dynamic graph property. That is, in each iteration, each subgraph \mathbb{G}_i evolves by obtaining or detecting nodes through a clustering procedure. To avoid an exhaustive computation of

Algorithm 2 Dynamic rank score tracking

Input: initial rank scores $\vec{r}_{PPR}^{(0)}$, initial residual $\vec{d}^{(0)}$.

Output: converged rank scores \vec{r}_{PPR} and converged residual \vec{d} .

```

1: for  $t = 1, 2, \dots$  do
2:   Pick the largest component  $\vec{d}_i^{(t-1)}$  in  $\vec{d}^{(t-1)}$ .
3:   while  $|\vec{d}_i^{(t)}| \geq \epsilon$  do
4:     Update  $\vec{r}_{PPR}^{(t)}$  and  $\vec{d}^{(t)}$ .
5:   end while
6: end for

```

the ranking procedure, our methods dynamically update the rank scores only for nodes that are (1) newly inserted into \mathbb{G}_i or (2) removed from \mathbb{G}_i after each clustering procedure.

We propose a *dynamic rank score tracking* that dynamically maintains rank scores of each time-evolving subgraph \mathbb{G}_i . Clearly, our proposed algorithms increase the approximation errors of rank scores because it partially computes rank scores and inserts/removes many nodes. Thus, to reduce the approximation errors, we employ a dynamic PPR computation technique [13] based on the Gauss-Southwell algorithm [2], [24]. Our proposed algorithms attempt to reduce the computation time without sacrificing the clustering quality of RankClus by dynamically updating only evolving nodes.

3.2 Dynamic Rank Score Tracking

To reduce the computation cost of the ranking procedure, our algorithm employs dynamic rank score tracking. Since each subgraph \mathbb{G}_i evolves after each clustering procedure, we update the rank scores of nodes that are newly inserted or removed from \mathbb{G}_i . Otherwise, our algorithm reuses the rank scores obtained before the clustering procedure. To perform the above updates efficiently, we adopt a dynamic PPR computation [13] based on the Gauss-Southwell algorithm [2], [24] into the RankClus.

Suppose, subgraph \mathbb{G}_i evolves as $\mathbb{G}_i^0 \rightarrow \mathbb{G}_i^1 \rightarrow \dots \mathbb{G}_i^t$ by iterating the clustering procedures, where \mathbb{G}_i^0 and \mathbb{G}_i^t are the initial subgraph and the subgraph after t -th iterations, respectively. The Gauss-Southwell algorithm maintains t -th rank score $\vec{r}_{PPR}^{(t)}$ of \mathbb{G}_i^t and its corresponding residual $\vec{d}^{(t)}$ as:

$$\vec{d}^{(t)} = (1 - \alpha)\vec{b} - (I - \alpha P)\vec{r}_{PPR}^{(t)}.$$

The goal of this algorithm is to minimize the residual $\vec{d}^{(t)}$, i.e., $\vec{d}^{(t)} \rightarrow \mathbf{0}$, since $\vec{r}_{PPR}^{(t)}$ converges when $\vec{d}^{(t)} = \mathbf{0}$. To minimize the residual $\vec{d}^{(t)}$ after the t -th clustering procedure, the algorithm picks the largest component $\vec{d}_i^{(t)}$ included in $\vec{d}^{(t)}$. Then it then computes $\vec{r}_{PPR}^{(t)}$ and $\vec{d}^{(t)}$ using following equations:

$$\begin{aligned} \vec{r}_{PPR}^{(t)} &= \vec{r}_{PPR}^{(t-1)} + \vec{d}_i^{(t-1)} e_i, \\ \vec{d}^{(t)} &= \vec{d}^{(t-1)} - \vec{d}_i^{(t-1)} e_i + \alpha \vec{d}_i^{(t-1)} P e_i, \end{aligned}$$

where e_i is a vector in which the i -th element is 1 and all other elements are 0. As shown in the above equations, the Gauss-Southwell algorithm propagates the largest residual to its neighbor nodes to reduce the approximation errors. Our algorithm continues the above updates until the largest residual satisfies $\vec{d}_i^{(t)} < \epsilon$. Consequently, this algorithm guarantees to have an error bound as $\|\vec{r}_{PPR}^* - \vec{r}_{PPR}^{(t)}\| \leq \frac{\epsilon}{(1-\alpha)}$, where \vec{r}_{PPR}^* is the ‘‘exact’’ rank scores. The pseudo-code of this algorithm is shown in Algorithm 2.

Algorithm 3 Proposed sequential algorithm**Input:** $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, K , and ϵ **Output:** X_i ($i = 1, 2, \dots, K$)

```

// Step 0: Initialization
1:  $t = 0$ .
2: Generate initial  $K$  clusters  $X_1^{(0)}, X_2^{(0)}, \dots, X_K^{(0)}$ .
3: repeat
  // Step 1: Dynamic Rank Score Tracking
4:   for  $i = 1$  to  $K$  do
5:     if  $t == 0$  then
6:       Compute  $\vec{r}_{PPR}(v, i)$  for all  $v \in X_i \cup Y$ .
7:       Compute residual  $\vec{d}^{(0)}$  and its corresponding  $\vec{r}^{(0)}$ .
8:     else
9:       Compute  $\vec{d}^{(t)}$  from  $\vec{d}^{(t-1)}$ .
10:      Compute  $r_{PPR}^{(t)}$  from  $r_{PPR}^{(t-1)}$ .
11:      Apply Algorithm 2 to  $r_{PPR}^{(t)}$  and  $\vec{d}^{(t)}$ .
12:    end if
13:    Compute  $\vec{r}_X^t(x, i) = \frac{\vec{r}_{PPR}(x, i)}{\sum_{x \in X_i} \vec{r}_{PPR}(x, i)}$  for all  $x \in X_i$ .
14:    Compute  $\vec{r}_Y^t(y, i) = \frac{\vec{r}_{PPR}(y, i)}{\sum_{y \in Y} \vec{r}_{PPR}(y, i)}$  for all  $y \in Y$ .
15:  end for
  // Step 2: Get new attribute
16:  for  $i = 1$  to  $K$  do
17:    for each  $x_j \in X_i$  do
18:      Estimate  $\pi_{j,i}$  by Definition 2.
19:    end for
20:    Determine a centroid vector  $\vec{s}_{X_i}$ .
21:  end for
  // Step 3: Assign  $x_j$  to cluster
22:  for each  $x_j \in X$  do
23:    for  $i = 1$  to  $K$  do
24:      Compute a distance  $D(j, k)$ .
25:    end for
26:    Obtain  $k_0 = \arg \min_k D(j, k)$ .
27:     $X_{k_0}^{(t+1)} = X_{k_0}^{(t)} \cup \{x_j\}$ .
28:  end for
29:   $t = t + 1$ .
30: until No clusters are updated.

```

3.3 Proposed Algorithm

As shown in Algorithm 3, our algorithm replaces the exhaustive ranking procedure in Algorithm 1 with the dynamic rank score tracking method (lines 4–15). In Step 1, if t is 0, $\vec{r}_{PPR}(v, i)$ is initialized for all nodes by the Personalized PageRank algorithm for each subgraph. Otherwise, our algorithm updates $\vec{r}_{PPR}(v, i)$ by Algorithm 2 to the rank scores, which avoids computing all nodes in each subgraph. After converging Algorithm 2, our method converts the rank scores into \vec{r}_X and \vec{r}_Y . Next, our algorithm moves to the clustering procedure (lines 16–28 in Algorithm 3). Here, the detail of the clustering procedure are omitted because same as those of RankClus shown in Algorithm 1.

3.4 Parallelization

For further improving the computational efficiency of Algorithm 3, we extend our algorithm to utilize multi-threading techniques on a manycore CPU. We employ thread-based parallelizations into loop-blocks that compute ranks of nodes. As shown as in the previous section, our proposed method performs the computation of rank scores for each clusters, i.e., X_1', X_2', \dots, X_K' . Since the computation of rank scores are independent for each cluster, we can output the same results as those of Algorithm 3 even if we parallelize Step 1 in Algorithm 3. Thus, we apply task-wise parallelization for the steps to reduce the computation time of Algorithm 3. Algorithm 4 shows the pseudocode of our multi-threading (parallel) algorithm, which is an extension of Al-

Algorithm 4 Parallel proposed algorithm**Input:** $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, K , and ϵ **Output:** X_i ($i = 1, 2, \dots, K$)

```

// Step 0: Initialization
1:  $t = 0$ .
2: Generate initial  $K$  clusters  $X_1^{(0)}, X_2^{(0)}, \dots, X_K^{(0)}$ .
3: repeat
  // Step 1: Dynamic Rank Score Tracking
4:   for  $i = 1$  to  $K$  do in thread-parallel do
5:     if  $t == 0$  then
6:       Compute  $\vec{r}_{PPR}(v, i)$  for all  $v \in X_i \cup Y$ .
7:       Compute residual  $\vec{d}^{(0)}$  and its corresponding  $\vec{r}^{(0)}$ .
8:     else
9:       Compute  $\vec{d}^{(t)}$  from  $\vec{d}^{(t-1)}$ .
10:      Compute  $r_{PPR}^{(t)}$  from  $r_{PPR}^{(t-1)}$ .
11:      Apply Algorithm 2 to  $r_{PPR}^{(t)}$  and  $\vec{d}^{(t)}$ .
12:    end if
13:    Compute  $\vec{r}_X^t(x, i) = \frac{\vec{r}_{PPR}(x, i)}{\sum_{x \in X_i} \vec{r}_{PPR}(x, i)}$  for all  $x \in X_i$ .
14:    Compute  $\vec{r}_Y^t(y, i) = \frac{\vec{r}_{PPR}(y, i)}{\sum_{y \in Y} \vec{r}_{PPR}(y, i)}$  for all  $y \in Y$ .
15:  end for
  // Step 2: Get new attribute
16:  for  $i = 1$  to  $K$  do
17:    for each  $x_j \in X_i$  do
18:      Estimate  $\pi_{j,i}$  by Definition 2.
19:    end for
20:    Determine a centroid vector  $\vec{s}_{X_i}$ .
21:  end for
  // Step 3: Assign  $x_j$  to cluster
22:  for each  $x_j \in X$  do
23:    for  $i = 1$  to  $K$  do
24:      Compute a distance  $D(j, k)$ .
25:    end for
26:    Obtain  $k_0 = \arg \min_k D(j, k)$ .
27:     $X_{k_0}^{(t+1)} = X_{k_0}^{(t)} \cup \{x_j\}$ .
28:  end for
29:   $t = t + 1$ .
30: until No clusters are updated.

```

gorithm 3. As we can see Algorithm 4, the workflow is the same as that of Algorithm 3. Specifically, (line 4) in Algorithm 4, we assign the ranking computation to a single thread.

4. Experimental Analysis

We experimentally analyzed the effectiveness of our algorithm to reduce the computation time of RankClus while keeping the clustering accuracy of bi-type information networks.

4.1 Setup

Algorithms: We compared the effectiveness of three methods:

- **Proposal:** Our proposed algorithm (Algorithm 3) that employs dynamic the rank score tracking method shown in Section 3. Unless otherwise stated, $\epsilon = 10^{-9}$ is the default setting of the Gauss-Southwell algorithm [2], [13], [24]. Unless otherwise stated, we used a single threaded algorithm.
- **RankClus:** The original algorithm (Algorithm 1) used to extract clusters from bi-type information networks [26].
- **Pruning:** The state-of-the-art method that reduces the computation time of RankClus by employing threshold-based pruning [29], [30]. Here the node-pruning parameters are set to the default settings shown in the literature [29].

We implemented the above three algorithms using C++11 and compiled them with gcc 8.2.0 with -O2 option. All experiments were conducted on a Linux server with a CPU (Intel Xeon E5-1620 3.50 GHz) and 128 GB main memory. All algorithms used

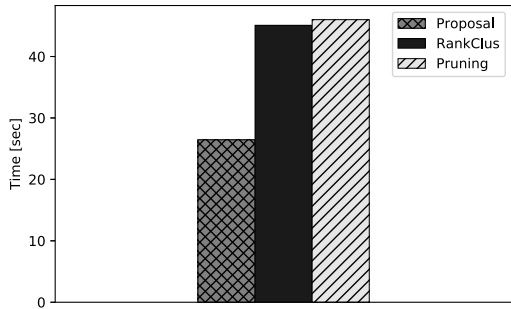


Fig. 3 Running times of each algorithm for Yahoo-msg.

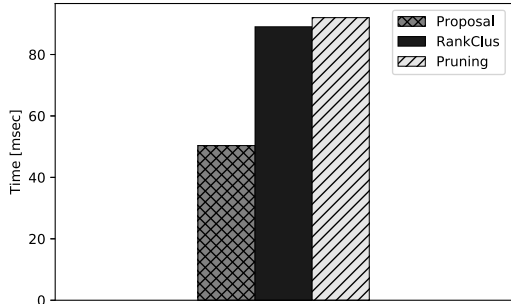


Fig. 4 Running times of each algorithm for DBLP.

$\alpha = 0.85$ for the PPR parameter in the ranking procedure.

Real-world datasets: We used two real-world bi-type information networks published in public data repositories.

- **DBLP [29]:** A bibliographic graph extracted from DBLP, which represents the relationships among 20 representative computer science conferences and 5,639 authors who have published more than two conference papers. This graph handles two types of relationships as edges: conference-author relationships and co-author relationships; In total, there are 95,516 edges in this dataset. We set the conferences and authors as the target type and attribute type, respectively.
- **Yahoo-msg [16]:** A social graph extracted from Yahoo Messenger in which users communicate with other users. This graph is composed of 100,001 users and 57 locations that are associated with the users. We set the locations and users as the target types and attribute types, respectively. This graph handles user-user relationships and user-location relationships. Once a user communicates with other users, they are linked the users by bi-directional edges. Additionally, users are linked with their representative locations based on their zip codes. As a result, this dataset has 6,359,436 edges.

Evaluation metric: We compared the accuracy using an information-theoretic metric, called normalized mutual information (NMI) [25]. Once two clustering results are obtained, NMI returns a score between 0 and 1. A score of 1 indicates the clusters are the same, while score of 0 means they are completely different.

4.2 Running Time Analysis

We evaluated the efficiency of our proposed algorithm by comparing running times of the above real-world bi-type information networks. In the evaluation, the number of clusters K was set to as $K = 4$ and $K = 10$ for DBLP and Yahoo-msg, respectively.

Figures 3 and 4 show the running time of each algorithm on

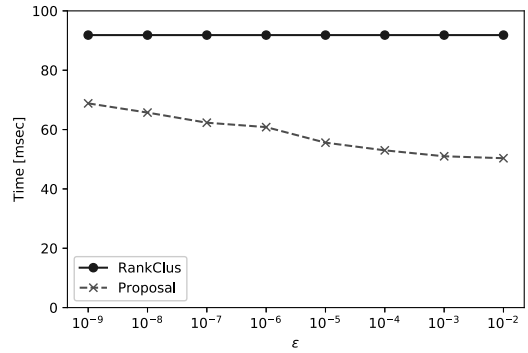


Fig. 5 Running times by varying ϵ on DBLP.

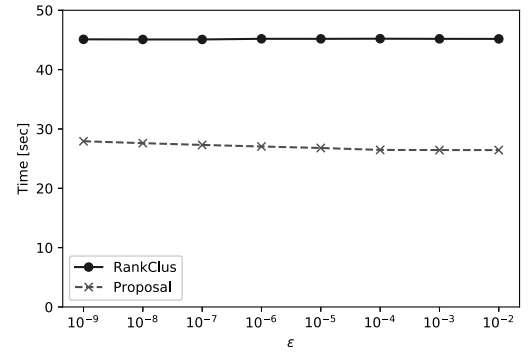


Fig. 6 Running times by varying ϵ on Yahoo-msg.

each dataset. Then we set $\epsilon = 10^{-9}$. Our proposed algorithm outperforms the other algorithms examined. Specifically, Proposal is up to twice as fast as the other algorithms, suggesting that our dynamic rank score tracking method mitigates the exhaustive computations incurred by the original RankClus framework. By contrast, the state-of-the-art algorithm (Pruning) increases the running times compared to RankClus because the real-world datasets shown in Section 4.1 have sparse connections even though Pruning typically fails to prune nodes on sparse graphs [29]. Hence, Proposal is superior to the state-of-the-art algorithm for real-world bi-type information networks. Hereafter, we omit the results of Pruning since it does not reduce the running times for sparse datasets (Figs. 3 and 4)

Next, we assessed the effect of the user-specified parameter ϵ of Proposal. We compared running times of Proposal by varying ϵ with those of RankClus. Figures 5 and 6 show the running times of DBLP and Yahoo-msg, respectively. We varied ϵ from 10^{-9} to 10^{-2} . Our proposed algorithm gradually reduces the running times as the ϵ value increases because our dynamic rank score tracking method only needs to update $\vec{r}_{PPR}^{(t)}$ and $\vec{d}^{(t)}$ until the largest residual satisfies $d_i^{(t)} < \epsilon$. Thus, Proposal terminates earlier for larger ϵ values.

We finally evaluated the impact of the number of clusters K on the running times. Figure 7 shows the runtimes when K was varied for the Yahoo-msg dataset. As K in Proposal increases, the speeding-up ratio increases because each subgraph \mathbb{G}_i does not drastically change its cluster members even if the K is large. In particular, if a subgraph \mathbb{G}_i does not have updates, our proposed algorithm can skip the ranking procedure for the subgraph, whereas the other algorithm must perform PPR on all subgraphs. Hence, the efficiency can be improved for larger K settings.

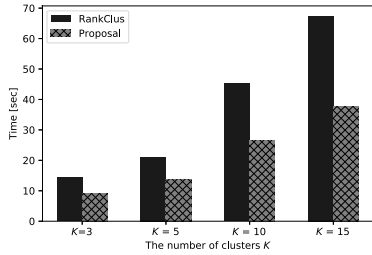


Fig. 7 Running times by varying K for Yahoo-msg.

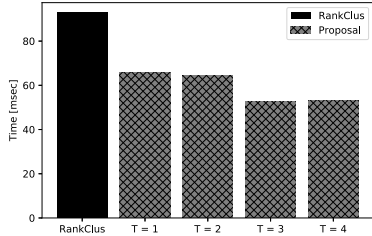


Fig. 8 Running times by varying T on DBLP.

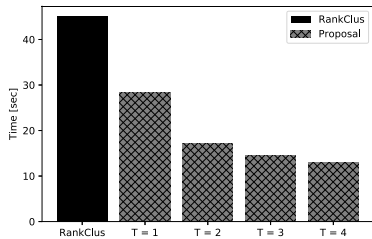


Fig. 9 Running times by varying T on Yahoo-msg.

4.2.1 Effectiveness of Our Multi-threading Approach

We here experimentally discuss the effectiveness of the parallelization techniques. In this evaluation, we compared the running time of RankClus with that of our parallel algorithm by varying the number of threads invoked in our proposed algorithm. **Figures 8 and 9** show the running times of DBLP and Yahoo-msg, respectively. We varied the number of threads T from 1 to 4 for testing the effectiveness of thread-based parallelization. Note that the parallel algorithm is equivalent to Algorithm 3 if we set $T = 1$. As we can see in Figs. 8 and 9, our proposed method shows faster clustering time compared with RankClus. Furthermore, we can observe from the figure that our parallel algorithm successfully reduces the running time by increasing the number of threads. Although the dynamic rank score tracking is a bottleneck of our proposed method, we can reduce its computational costs by introducing task-wise parallelization since the dynamic rank score tracking is independent among clusters. Specifically, when we set $T = 4$, our proposed method on Yahoo-msg dataset is almost 3.5 times faster than the original RankClus algorithm. These results indicate that our parallelization approach is effective in reducing the running time of RankClus.

4.3 Accuracy of Clustering Results

We assessed the accuracy of the clustering results produced by the proposed algorithm. In this evaluation, we measured the NMI scores between clusters extracted by Proposal and RankClus. That is, if NMI scores are close to 1, our proposed algorithm extracts almost the same results as RankClus. Herein we varied

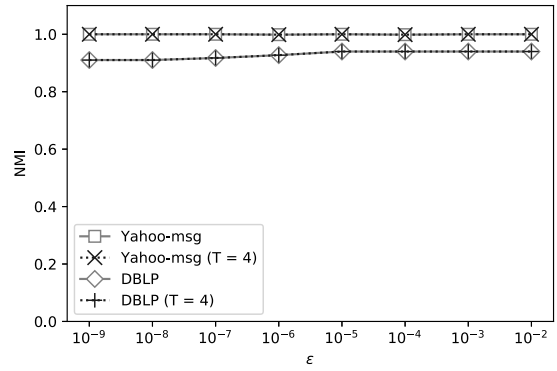


Fig. 10 NMI score of Proposal by varying ϵ . Although Proposal effectively reduces the running time (Section 4.2), it keeps significantly high NMI scores (≥ 0.9) for all ϵ settings.

the ϵ values of Proposal from 10^{-9} to 10^{-2} .

Figure 10 shows NMI scores of Proposal. Our proposed method shows high NMI values for all ϵ settings. As mentioned in Section 4.2.1, our parallel algorithm outputs the same NMI results as our non-parallel algorithm. In particular, our algorithm always shows NMI scores higher than 0.9, even though it has drastically reduced running times compared to RankClus. These results experimentally confirm the effectiveness of our proposed approaches on real-world bi-type information networks.

5. Related Work

Graph clustering is a fundamental graph analysis technique. Various algorithms have been proposed [14], [22] in recent decades. This section briefly reviews representative graph clustering algorithms.

Modularity clustering [3], [12], [19], [20] is a famous graph clustering algorithm. Modularity clustering finds sets of nodes with dense inner-group edges and sparse inter-group edges. Although modularity clustering can efficiently handle large-scale graphs, it can not fully reproduce the ground-truth clusters [8]. To overcome the low accuracy of modularity clustering algorithms, density-based graph clustering algorithms [23], [27], [28] have been proposed in recent years. Density-based graph clustering is an extension of DBSCAN [7], which is a traditional density-based clustering algorithm for multi-dimensional data objects. Similar to DBSCAN, the algorithm extracts clusters by measuring cluster densities, and thus they exclude sparsely connected nodes from clusters as hubs or outliers. Consequently, the density-based graph clustering can achieve a better clustering accuracy than that of modularity clustering.

Although the above approaches provide insight of overview hidden in a graph, they are not applicable to heterogeneous graphs such as the bi-type information networks that are tackled in this paper. This is because these approaches ignore types or attributes attached to nodes in the heterogeneous graphs. As a clustering method for heterogeneous graphs, Qi et al. [15] proposed a clustering algorithm based on heterogeneous Markov random fields. Also, Cruz et al. [6] presented a community detection algorithm by integrating multiple node attributes within a social network.

Recently, ranking-based clustering algorithms have achieved highly accurate clusters on bi-type information networks. Ex-

ample include RankClus and RankComplete [5]. The main concept behind ranking-based methods is to improve clustering accuracy by combining node ranking techniques [1], [17] and clustering procedures. These ranking-based clustering methods take advantage of rich clustering results and mutual improvements of clustering and ranking. However, as described in Section 2, these rank-based methods, especially RankClus, have high computational costs because they iteratively perform the ranking procedure for all nodes until clusters converge. Although Yamazaki et al. recently proposed a node-pruning algorithm for RankClus [29], [30], it can not handle large-scale bi-type information networks. As shown in Section 4.3, the running time increases if a graph is sparse, even though real-world bi-type information networks are generally sparse.

In this paper, we proposed an efficient algorithm for RankClus by introducing dynamic rank score tracking, which skips unnecessary rank score computations during the ranking procedure. As experimentally confirmed in Section 4, our proposed method achieves a faster clustering than our competitive algorithms without sacrificing the clustering quality compared to RankClus.

6. Conclusion

In this paper, we proposed two algorithms to improve the efficiency of RankClus algorithm for large-scale bi-type information networks. The first one is the dynamic rank score tracing algorithm that only computes the evolved nodes and their neighbor nodes so as to reduce their approximation errors in each iteration. In order to find evolved nodes, we focus on the dynamic graph property of RankClus framework, and adopt a dynamic Personalized PageRank computation based on the Gauss-Southwell method. The second one is a multi-threading method that is an extension of our dynamic rank score tracking algorithm. In the dynamic rank score tracking algorithm, we need perform the computations for all subgraph iteratively. To overcome the performance limitation, we employed a task-wise parallelization to speed up the dynamic rank score tracking in our parallel algorithm. Our extensive experiments using real-world datasets demonstrate our proposed method provides clusters almost twice as fast as competitive algorithms while keeping the clustering accuracy of the original RankClus algorithm. Furthermore, we experimentally confirmed that our parallel algorithm successfully reduces the running time of the rank score tracking method by increasing the number of thread invoked in the algorithm. The RankClus framework plays an important role in current and prospective Web-based system and applications in various disciplines. Our efficient algorithms will help to improve the effectiveness of future applications.

Acknowledgments This work was supported by JSPS KAKENHI Early-Career Scientists (Grant Number JP18K18057), JST ACT-I, and Interdisciplinary Computational Science Program in CCS, University of Tsukuba.

References

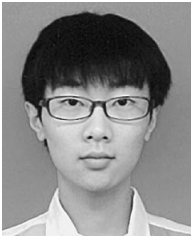
- [1] Balmin, A., Hristidis, V. and Papakonstantinou, Y.: ObjectRank: Authority-Based Keyword Search in Databases, *Proc. VLDB*, pp.564–575 (2004).
- [2] Berkhin, P.: Bookmark-coloring algorithm for personalized PageRank computing, *Internet Math.*, Vol.3, No.1, pp.41–62 (2006) (online), available from (<https://projecteuclid.org:443/euclid.im/1175266367>).
- [3] Blondel, V., Guillaume, J., Lambiotte, R. and Mech, E.: Fast Unfolding of Communities in Large Networks, *Journal of Statistical Mechanics: Theory and Experiment*, Vol.2008, No.10, p.P10008 (2008).
- [4] Brin, S. and Page, L.: The Anatomy of a Large-scale Hypertextual Web Search Engine, *Comput. Netw. ISDN Syst.*, pp.107–117 (1998).
- [5] Cao, L., Jin, X., Yin, Z., Del Pozo, A., Luo, J., Han, J. and Huang, T.S.: Rankkompete: Simultaneous ranking and clustering of information networks, *Neurocomputing*, pp.98–104 (2012).
- [6] Cruz, J.D., Bothorel, C. and Poulet, F.: Integrating Heterogeneous Information Within a Social Network for Detecting Communities, *Proc. 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM '13*, pp.1453–1454 (2013).
- [7] Ester, M., Kriegel, H.-P., Sander, J. and Xu, X.: A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, *Proc. 2nd International Conference on Knowledge Discovery and Data Mining*, pp.226–231 (1996).
- [8] Fortunato, S. and Barthelemy, M.: Resolution Limit in Community Detection, *Proc. National Academy of Sciences* (2007).
- [9] Jeh, G. and Widom, J.: Scaling Personalized Web Search, *Proc. 12th International Conference on World Wide Web, WWW2003*, pp.271–279 (2003).
- [10] Kleinberg, J.M.: Authoritative Sources in a Hyperlinked Environment, *J. ACM*, pp.604–632 (1999).
- [11] MacQueen, J.B.: Some Methods for Classification and Analysis of MultiVariate Observations, *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, Vol.1, pp.281–297, University of California Press (1967).
- [12] Newman, M.E.J. and Girvan, M.: Finding and Evaluating Community Structure in Networks, *Physical Review E*, Vol.69, 026113 (2004).
- [13] Ohsaka, N., Maehara, T. and Kawarabayashi, K.-I.: Efficient PageRank Tracking in Evolving Networks, *KDD '15*, pp.875–884 (2015).
- [14] Onizuka, M., Fujimori, T. and Shiokawa, H.: Graph Partitioning for Distributed Graph Processing, *Data Science and Engineering*, Vol.2, No.1, pp.94–105 (2017).
- [15] Qi, G.-J., Aggarwal, C.C. and Huang, T.S.: On Clustering Heterogeneous Social Media Objects with Outlier Links, *Proc. 5th ACM International Conference on Web Search and Data Mining, WSDM '12*, pp.553–562 (2012).
- [16] Rossi, R.A. and Ahmed, N.K.: The Network Data Repository with Interactive Graph Analytics and Visualization, *AAAI* (online), available from (<http://networkrepository.com>) (2015).
- [17] Sato, T., Shiokawa, H., Yamaguchi, Y. and Kitagawa, H.: FORank: Fast ObjectRank for Large Heterogeneous Graphs, *Companion Proc. Web Conference 2018*, pp.103–104 (2018).
- [18] Shi, J. and Malik, J.: Normalized Cuts and Image Segmentation, *IEEE Trans. Pattern Anal. Mach. Intell.*, pp.888–905 (2000).
- [19] Shiokawa, H., Amagasa, T. and Kitagawa, H.: Scaling Fine-grained Modularity Clustering for Massive Graphs, *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pp.4597–4604 (2019).
- [20] Shiokawa, H., Fujiwara, Y. and Onizuka, M.: Fast Algorithm for Modularity-based Graph Clustering, *Proc. 27th AAAI Conference on Artificial Intelligence*, pp.1170–1176 (2013).
- [21] Shiokawa, H., Fujiwara, Y. and Onizuka, M.: SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs, *Proc. Very Large Data Bases Endowment*, Vol.8, No.11, pp.1178–1189 (2015).
- [22] Shiokawa, H. and Onizuka, M.: *Scalable Graph Clustering and Its Applications*, pp.1–10, Springer New York (2017).
- [23] Shiokawa, H., Takahashi, T. and Kitagawa, H.: ScaleSCAN: Scalable Density-based Graph Clustering, *Proc. 29th International Conference on Database and Expert Systems Applications*, pp.18–34, DEXA (2018).
- [24] Spielman, D.A. and Teng, S.: A Local Clustering Algorithm for Massive Graphs and Its Application to Nearly Linear Time Graph Partitioning, *SIAM J. Comput.*, Vol.42, No.1, pp.1–26 (2013).
- [25] Strehl, A. and Ghosh, J.: Cluster Ensembles — A Knowledge Reuse Framework for Combining Multiple Partitions, *J. Mach. Learn. Res.*, pp.583–617 (2003).
- [26] Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H. and Wu, T.: RankClus: Integrating Clustering with Ranking for Heterogeneous Information Network Analysis, *EDBT '09*, pp.565–576 (2009).
- [27] Takahashi, T., Shiokawa, H. and Kitagawa, H.: SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors, *Proc. 2nd International Workshop on Network Data Analytics*, pp.6:1–6:7, NDA (2017).
- [28] Xu, X., Yuruk, N., Feng, Z. and Schweiger, T.A.J.: SCAN: A Structural Clustering Algorithm for Networks, *Proc. 13th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining (KDD), pp.824–833, ACM (2007).

- [29] Yamazaki, K., Sato, T., Shiokawa, H. and Kitagawa, H.: Fast Algorithm for Integrating Clustering with Ranking on Heterogeneous Graphs, *Proc. 20th International Conference on Information Integration and Web-based Applications & Services, iiWAS2018*, pp.24–32 (2018).
- [30] Yamazaki, K., Sato, T., Shiokawa, H. and Kitagawa, H.: Fast and Parallel Ranking-based Clustering for Heterogeneous Graphs, *Journal of Data Intelligence*, Vol.1, No.2, pp.137–158 (2019).



Kotaro Yamazaki is currently working as a Machine Learning Engineer at ZOZO Technologies, Inc. He received B.S., M.E., in engineering from University of Tsukuba in 2018 and 2020, respectively. His research interests include large-scale graph data analysis.



Shohei Matsugu received B.S. in engineering from University of Tsukuba in 2019. He is currently a Master student in University of Tsukuba. His research interests include large-scale graph data analysis.



Hiroaki Shiokawa is an Associate Professor at University of Tsukuba. He received B.S., M.E., and Ph.D. in engineering from University of Tsukuba in 2009, 2011 and 2015, respectively. From 2011 to 2015, he was a research scientist at Nippon Telegraph and Telephone Corporation, and he joined Center for Computational Sciences at University of Tsukuba in Nov. 2015. His current research interests include database systems, data engineering, data mining, and graph data management.



Hiroyuki Kitagawa received his B.Sc. degree in physics and his M.Sc. and Dr. Sc. degrees in computer science, all from the University of Tokyo. He is currently a full professor at Center for Computational Sciences and Center for Artificial Intelligence Research, University of Tsukuba. His research interests include databases, data integration, data mining, information retrieval, and data engineering applications. He served as President of the Database Society of Japan from 2014 to 2016. He is an IEICE Fellow, an IPSJ Fellow, an Associate Member of the Science Council of Japan, and a member of ACM, IEEE, JSST.

(Editor in Charge: *Ikki Ohmukai*)