Fast and Parallel Dynamic Ranking-based Clustering for Heterogeneous Graphs

Kotaro YAMAZAKI†, Shohei MATSUGU†, Hiroaki SHIOKAWA††, and Hiroyuki KITAGAWA††

† Graduate School of System and Information Engineering,
University of Tsukuba Tsukuba, Ibaraki 305–8573, Japan
†† Center for Computational Sciences,
University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
E-mail: †{kyamazaki,matsugu}@kde.cs.tsukuba.ac.jp, ††{shiokawa,kitagawa}@cs.tsukuba.ac.jp

Abstract The RankClus framework accurately performs clustering using ranking-based graph clustering techniques. It integrates graph ranking algorithms into graph clustering procedures to improve the clustering quality. However, this integration incurs a high computational cost since RankClus repeatedly computes the ranking algorithm for all nodes until the clusters converges. To overcome this limitation, we present a novel RankClus algorithm that reduces the running time. By dynamically updating ranking results, our proposal reduces the number of computed nodes and edges. For further improving the efficiency, we also present a parallel implementation of our proposed algorithm by using thread-based parallelization. We experimentally verify using real-world datasets that our proposed methods successfully reduces the running time while maintaining the clustering quality of RankClus.
Key words RankClus, Graph Mining, Clustering

1 Introduction

How can clusters be computed within a short computation time for large bi-type information networks? 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 [10]. 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 [9], modularity clustering [10], [11], and density-based methods [12], [13], [16] 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 [15], 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 [15]. 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 [3] or HITS [5] 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 highly accurate clusters included in the target-type

nodes [15].

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 [6] 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 [17], [18] 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.

To tackle this problem, we present a novel algorithm for RankClus that effectively reduces the exhaustive computational costs without degrading the clustering quality.

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 $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ 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 collaboration 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:

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

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

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

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

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

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

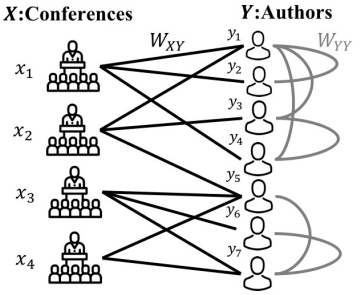

Figure 1 Example of a bi-type information network: This graph has two types of nodes X and Y, which represent sets of conferences and authors, respectively. If author y_j has published at least one paper at conference x_i, there is a link between node x_i and y_j. Additionally, two authors y_i and y_j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

2.2 RankClus Framework

The RankClus framework [15] 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

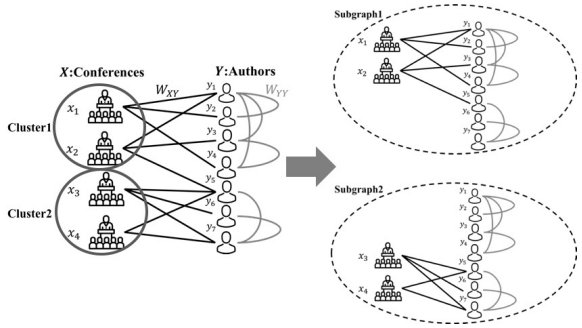

Figure 2 Example of subgraph construction using Figure 1: In this example, we set $K = 2$ and partition $X = \{x_1, x_2, x_3, x_4\}$ into two subsets $\{x_1, x_2\}$ and $\{x_3, x_4\}$. Then a subgraph is constructed for each subset.

Algorithm 1 RankClus

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

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

// Step 0: Initialization
$t = 0$.
Generate initial K clusters $X_1^{(0)}, X_2^{(0)}, \ldots, X_K^{(0)}$.
repeat
  // Step 1: Ranking for each cluster
  Construct subgraphs $\mathbb{G}_i(t)$ from $X_i(t)$, and Y.
  for $i = 1$ to K **do**
    for each $x_j \in X_i$ and $y_s \in Y$ **do**
      Compute rank score $\vec{r}_X(j, i)$ and $\vec{r}_Y(s, i)$.
    end for
  end for
  // Step 2: Get new attribute
  for $i = 1$ to K **do**
    for each $x_j \in X_i$ **do**
      Estimate $\pi_{j,i}$ by Definition 2.
    end for
    Determine a centroid vector $\vec{s}_{X_i}$.
  end for
  // Step 3: Assign x_j to cluster
  for each $x_j \in X$ **do**
    for $i = 1$ to K **do**
      Compute a distance $D(j, k)$
    end for
    Obtain $k_0 = \arg\min_k D(j, k)$.
    $X_{k_0}^{(t+1)} = X_{k_0}^{(t+1)} \cup \{x_j\}$.
  end for
  $t = t + 1$.
until No clusters are updated.

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 Algorithm1. Algorithm 1 takes bi-type information network $\mathbb{G}$ and a number of cluster K as inputs. The RankClus framework 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 ($1 \leqq i \leqq K$). 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 $\mathbb{E}_i = \{\langle o_i, o_j \rangle | o_i, o_j \in X_i \cup Y\}$. 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.

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 [4]. 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 procedure. 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

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

After that, the framework constructs K-dimensional feature vector $s_{x_i} = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,K}\}$ 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:

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

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 x_{X_i} 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 shown in the previous section 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 in each clustering procedure. 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 algorithms are to handle the above dynamic graph property of the RankClus framework. 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 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 [7] based on the Gauss-Southwell algorithm [1], [2]. 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 [7] based on the Gauss-Southwell algorithm [1], [2] into the RankClus framework.

Suppose, subgraph $\mathbb{G}_i$ evolves as $\mathbb{G}_i^0 \rightarrow \mathbb{G}_i^1 \rightarrow \ldots \mathbb{G}_i^t$ 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 $||r_{PPR}^{*} - r_{PPR}^{(t)}|| \leqq \frac{\epsilon}{(1-\alpha)}$,

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

Algorithm 3 Proposed sequential algorithm

Input: $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, K, and ϵ
Output: X_i $(i = 1, 2, \cdots, K)$

// Step 0: Initialization
1: $t = 0$.
2: Generate initial K clusters $X_1^{(0)}, X_2^{(0)}, \ldots, X_K^{(0)}$.
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 $r\vec{P}\vec{P}R^{(t)}$ from $r\vec{P}\vec{P}R^{(t-1)}$.
11: Apply Algorithm 2 to $r\vec{P}\vec{P}R^{(t)}$ and $\vec{d}^{(t)}$.
12: **end if**
13: Compute $r\vec{X}(x, i) = \frac{\vec{r}_{PPR}(x,i)}{\sum_{x \in X_k} \vec{r}_{PPR}(x,i)}$ for all $x \in X_i$.
14: Compute $r\vec{Y}(y, i) = \frac{\vec{r}_{PPR}(y,i)}{\sum_{y \in Y} \vec{r}_{PPR}(y,i)}$ 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: $X_{k_0}^{(t+1)} = X_{k_0}^{(t+1)} \cup \{x_j\}$.
28: **end for**
29: $t = t + 1$.
30: **until** No clusters are updated.

where $\vec{r}_{PPR}$ is the "exact" rank scores. The pseudo-code of this algorithm is shown in Algorithm 2.

3.3 Proposed Algorithm

As shown in Algorithm 3, our proposed 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 proposed 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.,* $X_1^t, X_2^t, \ldots, X_K^t$. 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 Algorithm3. Algorithm 4 shows the pseudocode of our multi-threading (parallel) algorithm, which is an extension of Algorithm 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 proposed algorithm to reduce the computation time of RankClus while keeping the clustering accuracy of real-world 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. Unless otherwise stated, $\epsilon = 10^{-9}$ is the default setting of the Gauss-Southwell algorithm [1], [2], [7].Unless otherwise stated, we used a single threaded algorithm.
- **RankClus:** The original algorithm(Algorithm 1) used to extract clusters from bi-type information networks [15].
- **Pruning:** The state-of-the-art method that reduces the computation time of RankClus by employing threshold-based pruning [17], [18]. Here the node-pruning parameters are set to the default settings shown in the literature [17].

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

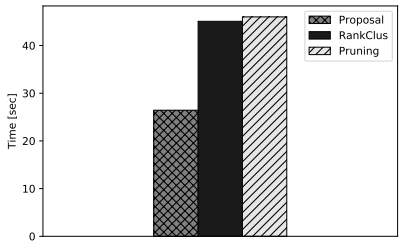

Figure 3 Running times of each algorithm for Yahoo-msg.

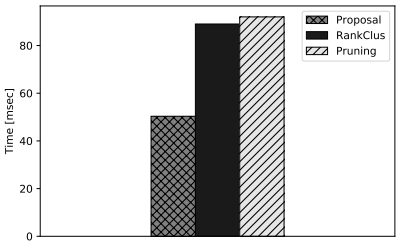

Figure 4 Running times of each algorithm for DBLP.

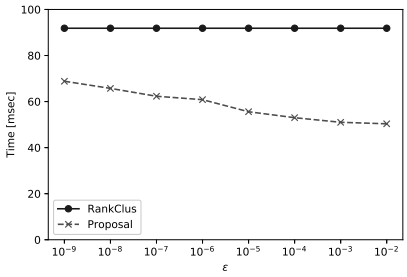

Figure 5 Running times by varying ϵ on DBLP.

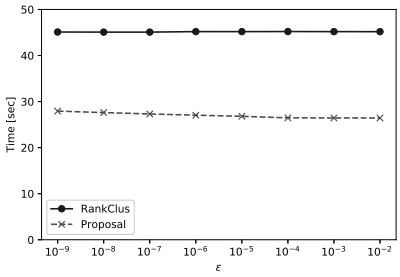

Figure 6 Running times by varying ϵ on Yahoo-msg.

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

- **DBLP [17]:** 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 [8]:** 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 are 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 clustering accuracy using an information-theoric metric, called normalized mutual information (NMI) [14]. 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 4 and 3 shows the running time of each algorithm on each dataset. For Proposal, $\epsilon = 10^{-9}$. 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 [17]. 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 (Figures. 4 and 3)

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 $\vec{d}_i^{(t)} < \epsilon$. 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 sub-

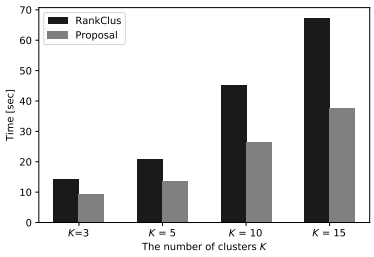

Figure 7 Running times by varying K for Yahoo-msg.

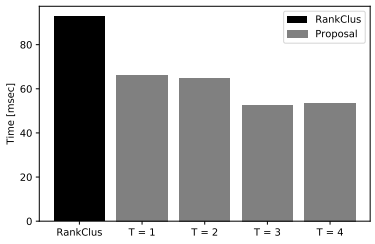

Figure 8 Running times by varying T on DBLP.

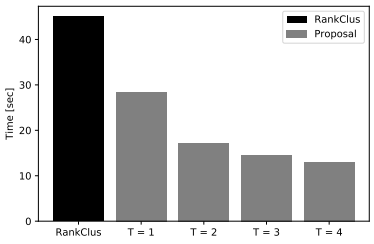

Figure 9 Running times by varying T on Yahoo-msg.

Algorithm 4 Parallel proposed algorithm

Input: $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, K, and ϵ
Output: X_i $(i = 1, 2, \cdots, K)$

// Step 0: Initialization
$t = 0$.
Generate initial K clusters $X_1^{(0)}, X_2^{(0)}, \ldots, X_K^{(0)}$.
repeat
  // Step 1: Dynamic Rank Score Tracking
  for $i = 1$ to K **do in thread-parallel do**
    if t == 0 **then**
      Compute $\vec{r}_{PPR}(v, i)$ for all $v \in X_i \cup Y$.
      Compute residual $\vec{d}^{(0)}$ and its corresponding $\vec{r}^{(0)}$.
    else
      Compute $\vec{d}^{(t)}$ from $\vec{d}^{(t-1)}$.
      Compute $r\vec{PPR}^{(t)}$ from $r\vec{PPR}^{(t-1)}$.
      Apply Algorithm 2 to $r\vec{PPR}^{(t)}$ and $\vec{d}^{(t)}$.
    end if
    Compute $r\vec{X}(x, i) = \frac{\vec{r}_{PPR}(x,i)}{\sum_{x \in X_k} \vec{r}_{PPR}(x,i)}$ for all $x \in X_i$.
    Compute $r\vec{Y}(y, i) = \frac{\vec{r}_{PPR}(y,i)}{\sum_{y \in Y} \vec{r}_{PPR}(y,i)}$ for all $y \in Y$.
  end for
  // Step 2: Get new attribute
  for $i = 1$ to K **do**
    for each $x_j \in X_i$ **do**
      Estimate $\pi_{j,i}$ by Definition 2.
    end for
    Determine a centroid vector $\vec{s}_{X_i}$.
  end for
  // Step 3: Assign x_j to cluster
  for each $x_j \in X$ **do**
    for $i = 1$ to K **do**
      Compute a distance $D(j, k)$.
    end for
    Obtain $k_0 = \arg\min_k D(j, k)$.
    $X_{k_0}^{(t+1)} = X_{k_0}^{(t+1)} \cup \{x_j\}$.
  end for
  $t = t + 1$.
until No clusters are updated.

graph $\mathbb{G}_i$ 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. Consequently, the efficiency can be improved for larger K settings.

4.2.1 Effectiveness of our multi-threading approach

We here experimentally discuss the effectiveness of the parallelization techniques shown in Algorithm 4. 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 Figures 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. 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. Herein we varied 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 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

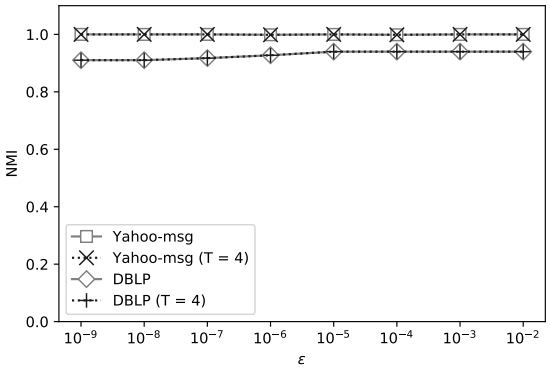

Figure 10 NMI score of Proposal by varying ϵ. Although Proposal effectively reduces the running time (Section 4.2), it keeps significantly high NMI scores ($\geqq 0.9$) for all ϵ settings.

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 dynaimc 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 parallelizaiton 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.

Acknowledgment

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] Daniel A. Spielman andShang-Hua Teng. A Local Clustering Algorithm for Massive Graphs and its Application to Nearly Linear Time Graph Partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

[2] Pavel Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math.*, 3(1):41–62, 2006.

[3] Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst.*, pages 107–117, 1998.

[4] Glen Jeh and Jennifer Widom. Scaling Personalized Web Search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW2003, pages 271–279, 2003.

[5] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, pages 604–632, 1999.

[6] J. B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[7] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient Pagerank Tracking in Evolving Networks. KDD '15, pages 875–884, 2015.

[8] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[9] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 888–905, 2000.

[10] Hiroaki Shiokawa, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Scaling Fine-grained Modularity Clustering for Massive Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 4597–4604, 2019.

[11] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pages 1170–1176, 2013.

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

[13] Hiroaki Shiokawa, Tomokatsu Takahashi, and Hiroyuki Kitagawa. ScaleSCAN: Scalable Density-based Graph Clustering. In *Proceedings of the 29th International Conference on Database and Expert Systems Applications*, DEXA, pages 18–34, 2018.

[14] Alexander Strehl and Joydeep Ghosh. Cluster Ensembles — a Knowledge Reuse Framework for Combining Multiple Partitions. *J. Mach. Learn. Res.*, pages 583–617, 2003.

[15] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. RankClus: Integrating Clustering with Ranking for Heterogeneous Information Network Analysis. EDBT '09, pages 565–576, 2009.

[16] Tomokatsu Takahashi, Hiroaki Shiokawa, and Hiroyuki Kitagawa. SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors. In *Proceedings of the 2nd International Workshop on Network Data Analytics*, NDA, pages 6:1–6:7, New York, NY, USA, 2017.

[17] Kotaro Yamazaki, Tomoki Sato, Hiroaki Shiokawa, and Hiroyuki Kitagawa. Fast Algorithm for Integrating Clustering with Ranking on Heterogeneous Graphs. In *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, iiWAS2018, pages 24–32, 2018.

[18] Kotaro Yamazaki, Tomoki Sato, Hiroaki Shiokawa, and Hiroyuki Kitagawa. Fast and Parallel Ranking-based Clustering for Heterogeneous Graphs. *Journal of Data Intelligence*, 1(2):137–158, 6 2019.