Journal of Data Intelligence, Vol. 1, No. 2 (2019) 137–158

FAST AND PARALLEL RANKING-BASED CLUSTERING FOR HETEROGENEROUS GRAPHS

KOTARO YAMAZAKI
Graduate School of SIE, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
k_yamazaki@kde.cs.tsukuba.ac.jp

TOMOKI SATO
Graduate School of SIE, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
t.sato@kde.cs.tsukuba.ac.jp

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

HIROYUKI KITAGAWA
Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
kitagawa@cs.tsukuba.ac.jp

The demands for graph data analysis methods are increasing. RankClus is a framework to extract clusters by integrating clustering and ranking on heterogeneous graphs; it enhances the clustering results by alternately updates the results of clustering and ranking for the better understanding of the clusters. However, RankClus is computationally expensive if a graph is large since it needs to iterate both clustering and ranking for all nodes. In this paper, to address this problem, we propose a novel fast RankClus algorithm for heterogeneous graphs. To speed up the entire procedure of RankClus, our proposed algorithm reduces the computational cost of the ranking process in each iteration. Our proposal measures how each node affects the clustering result; if it is not significant, we prune the node. Furthermore, we also present a parallel algorithm by extending our proposed algorithm by fully exploiting a modern manycore CPU. As a result, our extensive evaluations clarified that our fast and parallel algorithms drastically cut off the computation time of the original algorithm RancClus.

Keywords: Graph, Clustering, Ranking

1. Introduction

Graphs are one of the most fundamental data structures that represent schema-less and complicated relationships among data entities. They are now widely used not only in the Web-based applications and systems, but also in the various application domains including computer graphics, biological analysis, healthcare, and so on. As the graphs are increasing, graph analysis becomes essential data mining tools in the Web communities. *Ranking* and *clustering* are the most representative graph mining methods, and they play essential roles in

Table 1. Semantic clusters of the given conference list.

DB	SIGMOD, VLDB, PODS, ICDT, EDBT
IR	SIGIR, TREC, ECIR, JCDL, ECDL

Table 2. Top-10 conference ranks by using Table 1.

Rank	Conference
1	SIGMOD
2	VLDB
3	EDBT
4	SIGIR
5	PODS
6	TREC
7	ECIR
8	ICDT
9	JCDL
10	ECDL

the many applications [1]. The ranking, such as PageRank [2], HITS [3] and ObjectRank [4], is a method to evaluate the importance of each node in a graph based on an arbitrary ranking function. On the other hand, the clustering, *e.g.,* modularity-based [5, 6] and density-based algorithms [7, 8], groups nodes based on some proximity metrics so that similar nodes are assigned in the same group, and dissimilar nodes should belong to the different groups.

The clustering and the ranking are usually used independently; however, they often fail to capture a better understanding of the real-world datasets; the clustering without considering the ranking causes inappropriate or sometimes slightly biased analytical results, and vice versa. Here, we show an example of the inappropriate analytical results on the real bibliographic dataset due to independent use of the ranking and the clustering:

Example 1 (The ranking without considering the clustering)
Suppose that we have a conference relationship network taken from DBLP bibliographic database[a]*; nodes represent the ten conferences, i.e., SIGMOD, SIGIR, VLDB, TREC, PODS, ECIR, ICDT, JCDL, EDBT, and ECDL; and edges show citation relationships among papers published in the conferences. Note that, as shown in Table 1, the conferences are semantically separated into two clusters, i.e., DB cluster and IR cluster, but we do not know this clustering result.*

By using the conference relationship network, we applied the traditional graph ranking algorithm, e.g., PageRank [2]. Table 2 shows the ranking result. As we can see from Table 2, the DB conferences and IR conferences are intermingled in the ranks. This is because that the ranking algorithm, which we used, did not take care of the conference clusters shown in Table 1. This result is not helpful for a better understanding of the relationships of inner/inter research domains.

As shown in Example 1, the independent use of the clustering and the ranking fails to capture how large impacts each node have for each domain (cluster). Thus, it is a significant problem to explore how to integrate the clustering and the ranking methods.

[a] `http://dblp.uni-trier.de/`

Table 3. Top-5 ranks in DB.

Rank	Conf.
1	SIGMOD
2	VLDB
3	EDBT
4	PODS
5	ICDT

Table 4. Top-5 ranks in IR.

Rank	Conf.
1	SIGIR
2	TREC
3	ECIR
4	JCDL
5	ECDL

To address the above problem, Sun *et al.,* proposed RankClus algorithm [9] that successfully integrates the clustering and the ranking on the heterogeneous graphs. By performing the clustering and the ranking one after another, RankClus achieves better clustering and ranking results than the independent use of them. Specifically, RankClus first takes randomly partitioned clusters, and then it computes how high rank each node has for each cluster by performing the authority-based ranking algorithm [3]. After that RankClus *re*-construct clusters based on the assumption that a node has a higher rank score corresponding to an appropriate cluster; otherwise, it has a lower rank score. By following the above assumption, RankClus performs the traditional clustering [10, 11] by using how high rank scores each node has for the clusters. By performing ranking and clustering iteratively, the qualities of ranking and clustering are mutually enhanced. As a result, RankClus successfully achieves accurate and efficient graph data analysis compared with competitive algorithms. Here, we show an example of how RankClus can improve the analysis results on the dataset used in Example 1.

Example 2 (RankClus results using the ten conference network.)
Again, suppose that we use the same ten conference network dataset described in Example 1. By performing RankClus algorithm on the dataset, we can obtain the ranking results shown in Table 3 and Table 4 automatically. As we can see from the results, RankClus handles how substantial impacts each node have for each domain (cluster). Moreover, assigned ranks lead us to a better understanding of the relationships of the nodes in each cluster. Thus, by using RankClus, we can obtain fine-grained results from the complex datasets.

Although RankClus is useful to enhance graph analysis results, it is, however, computationally expensive since RankClus needs to iterate both the ranking and the clustering until clusters do not change significantly. Notably, in each ranking procedure, RankClus needs to generate subgraphs as many as the number of clusters, and it then needs to iteratively perform the ranking procedure for all nodes included in each subgraph until all rank scores converge. Clearly, these procedures incur high computational costs, and thus it is difficult to apply RankClust to the heterogeneous graphs.

To overcome the performance limitation in RankClus, we present an efficient algorithm [12] for speeding-up RankClus on large-scale heterogeneous graphs. To speed up the entire procedure of RankClus, we focused on the computational costs of the ranking procedure since it is the primary bottleneck in RankClus. Our proposed algorithm attempts to reduce the runtimes of the ranking procedure by incrementally pruning nodes that do not significantly affect the final ranking and clustering results.

Contribution: In summary, our contributions in this paper are shown as follows:

- We first derive a novel measure, named *change rate of rank score*, that evaluates how the

rank score increase/decrease compared to that in the previous iteration; if a node has a low change rate of the rank score, we determine that the node not have a significant impact on the final ranking and clustering results (Section 3.2).

- Then, by using the change rate of the rank score, we present a fast algorithm that approximates the RankClus results within short runtime (Section 3.3).
- For further improving the clustering speed, we extend our algorithm so that it fully utilizes thread-based parallelization on a modern manycore processor (Section 3.4).
- Finally, we conducted experiments on the various datasets, and we experimentally confirmed that our fast and parallel algorithms achieve faster analysis than the original RankClus algorithm while keeping the high accuracy (Section 4).

Our extensive evaluations showed that our proposed method without parallelization reduces more than 30% of runtimes of the original RankClus algorithm [9] while keeping its clustering qualities. Also, our multi-threading algorithm further cuts off at most 22% of the computation time consumed by our proposed algorithm. Even though RankClus is effective in enhancing various applications, it has been challenging to apply RankClus to large datasets due to its performance limitations. However, by providing our efficient approaches, our algorithms will help to improve the effectiveness of a broader range of applications.

Organization: The rest of paper is organized as follows: In Section 2, we briefly describe the backgrounds of this paper. In Section 3, we present our fast and parallel algorithms, and we then report the results of our experimental evaluations in Section 4. After that, in Section 5, we review the related work, and we finally conclude this paper in Section 6.

2. PRELIMINARIES

In this section, we define the notations and introduce the background of this paper. Table 5 shows the notations and their definitions used in this paper. We first show the data model of RankClus in Section 2.1 that is followed by the full descriptions of RankClus.

2.1. *Data Model*

We briefly introduce the data model of RankClus.

2.1.1. *Bi-type Information Network:*

RankClus takes a *bi-type information network* as an input data model that is a popular data model among various heterogeneous graphs. The formal definition of the bi-type information network is shown as follows:

Definition 1 (Bi-type Information Network) *Given two node sets* $X = \{x_1, x_2, \ldots, x_m\}$ *and* $Y = \{y_1, y_2, \ldots, y_n\}$*, where* m *and* n *are the numbers of nodes in* X *and* Y*, respectively; graph* $\mathbb{G}\langle\mathbb{V}, \mathbb{E}\rangle$ *is called a Bi-type Information Network on types* X *and* Y*.* $\mathbb{V}$ *and* $\mathbb{E}$ *are a node set* $\mathbb{V}(\mathbb{G}) = X \cup Y$ *and an edge set* $\mathbb{E}(\mathbb{G}) = \{\langle o_i, o_j\rangle | o_i, o_j \in X \cup Y\}$ *in the graph, respectively.*

Let w_{o_i,o_j} be the weight of edge $\langle o_i, o_j\rangle$, we denote the adjacency matrix of edges by $W_{(m+n)\times(m+n)} = \{w_{o_i,o_j}\}$. For convenience, we decompose the adjacency matrix into four

Table 5. Definitions of main symbols.

Notation	Definition
$\mathbb{G}$	A given graph.
$\mathbb{V}$	A set of nodes included in $\mathbb{G}$.
$\mathbb{E}$	A set of edges in $\mathbb{G}$.
$\mathbb{G}_i$	A subgraph given by $\mathbb{X}_i$.
$\mathbb{V}_i$	A set of nodes included in $\mathbb{G}_i$.
$\mathbb{E}_i$	A set of edges included in $\mathbb{G}_i$.
$\mathbb{P}_k$	A set of prunable nodes included in $\mathbb{G}_k$.
X_k	X A set of nodes included in cluster k.
X	A set of target type nodes.
Y	A set of attribute type nodes.
N	A number of nodes included in $\mathbb{V}$.
m	A number of nodes included in X.
n	A number of nodes included in Y.
K	A number of clusters.
T	A number of threads invoked in our parallel algorithm
W	$(m+n) \times (m+n)$ adjacency matrix.
$\vec{r}_X$	$m \times 1$ rank score vector of target type node set X.
$\vec{r}_Y$	$n \times 1$ rank score vector of attribute type node set Y.
$\pi_{i,k}$	Posteriori probability of node x_i when it belongs to cluster k.
$D(i,k)$	Distance between node x_i and cluster k.
$R^{(t)}(j,k)$	Change ratio of node x_j for subgraph $\mathbb{G}_k$ in the t-th iteration.
λ_R	Change-ratio parameter.
λ_I	Stability parameter.

blocks: W_{XX}, W_{XY}, W_{YX} and W_{YY}. Each block indicates a subgraph of the given bi-type information network among subscript types. W thus can be written as follows:

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

Figure 1 is an example of the bi-type information network that represents a conference-author collaboration network. Let X and Y be sets of conferences and authors, respectively; we can introduce the matrix W_{XY} as follows:

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

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

Similarly, W_{YY} is define as follows:

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

where a_{ij} is the number of papers that the author y_i has written with author y_j. W_{YX} is equal to W_{XY}^T because the relationship between conferences and authors is symmetric. Since there is no connection between conferences, $W_{XX} = O$.

2.1.2. *Target type and Attribute type:*

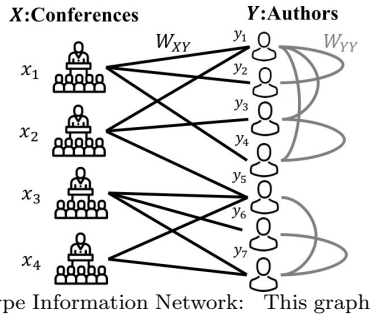

Fig. 1. Example of the Bi-type Information Network: This graph represents a conference-author relationship network. If author y_i has ever published a papers on conference x_i, we links between x_i and y_i. Also, we link between y_i and y_j if y_i has been a co-author of y_j, and vice versa.

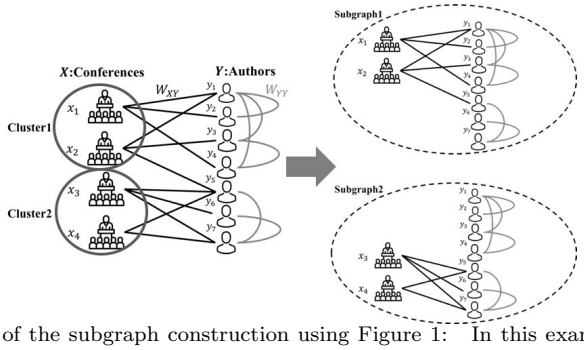

Fig. 2. Example of the subgraph construction using Figure 1: In this example, we set $K = 2$ and partition $X = \{x_1, x_2, x_3, x_4\}$ into two subsets $\{x_1, x_2\}$ and $\{x_3, x_4\}$. Then, we construct a subgraph for each subset.

RankClus defines two types of nodes, *target type* and *attribute type*, on X and Y; RankClus performs clustering *only* for the target type nodes, and it uses attribute type nodes as support information for the clustering. Recall the example of the bi-type information network shown in Figure 1. In Figure 1, RankClus clusters groups of conferences based on W_{XY} and W_{YY} if we set X and Y as target type and attribute type, respectively.

2.2. *RankClus Algorithm*

The goal of RankClus is to determine clusters of the target type nodes by using the attribute type nodes. In order to improve the clustering quality, RankClus iteratively performs the clustering and the ranking one after another. In this section, we briefly introduce the algorithm of RankClus.

In RankClus algorithm, we have the following observation: *"the node tends to have a high rank score only for the clusters what the node belongs."* By following this observation, RankClus constructs a feature vector from the rank scores for each node. Specifically, if we have K clusters, each node can take K rank scores in the algorithm, and thus RankClus constructs K-dimensional vector for each node by using the K rank scores. After obtaining the feature vectors, RankClus performs the clustering procedure. Briefly, the workflow of RankClus algorithm is shown in the below:

(1) Randomly assign initial K clusters for the target type nodes X, and constructs K

subgraphs.

(2) Computes rank scores of all nodes in X and Y for each subgraph.

(3) Construct K dimensional feature vector for each target type node by using the rank scores obtained in (2).

(4) Update the cluster assignments of the target type nodes by using the K dimensional vector obtained in (3).

(5) Repeat (2) to (4) until the clusters converge.

The workflow of RankClus algorithm can be divided into two parts: ranking part, *i.e.,* (1) and (2), and clustering part, *i.e.,* (3) and (4). We detail the ranking part and the clustering part in the Section 2.2.1 and Section 2.2.2 , respectively.

2.2.1. *Ranking part*

The goal of the ranking part is to determine the rank score of each node in $X \cup Y$ for each corresponding clusters.

RankClus first randomly partitions the target type nodes X into K clusters, and it then constructs K subgraphs. By letting X_i be the subset of X included in the i-th cluster ($1 \leq i \leq K$), each of the subgraphs is defined as $\mathbb{G}_i = \langle \mathbb{V}_i, \mathbb{E}_i \rangle$, where $\mathbb{V}_i = \{X_i \cup Y\}$ and $\mathbb{E}_i = \{\langle o_i, o_j \rangle | o_i, o_j \in X_i \cup Y\rangle\}$; clearly, we have $\mathbb{G} = \bigcup_i^K \mathbb{G}_i$. Figure 2 shows an example of the cluster assignment. In this example, we partition X into two clusters, *i.e.,* $\{x_1, x_2\}$ and $\{x_3, x_4\}$, and then construct corresponding subgraphs.

After constructing the subgraphs, RankClus computes the rank score of each node in $X \cup Y$ for each subgraph $\mathbb{G}_i$ based on authority-based ranking function [3]. Let $\vec{r}_X(x)$ and $\vec{r}_Y(y)$ be the rank score of node x in X and node y in Y, respectively. For each subgraph $\mathbb{G}_i$, RankClus obtains the rank scores based on the authority-based ranking function by using the following definitions:

Definition 2 (Authority Ranking) *Let $\vec{r}_X(i)$ and $\vec{r}_Y(i)$ be the rank score of $x_i \in X$ and $y_i \in Y$ for a subgraph $\mathbb{G}_k$, respectively; $\vec{r}_X(i,k)$ and $\vec{r}_Y(i,k)$ is given from following equations.*

$$\vec{r}_X(i,k) = \alpha \sum_{j=1}^{m} W_{XY}(i,j)\vec{r}_Y(j,k),$$

$$\vec{r}_Y(i,k) = \alpha \sum_{j=1}^{m} W_{XY}(i,j)\vec{r}_X(j,k) + (1-\alpha)\sum_{j=1}^{n} W_{YY}(i,j)\vec{r}_Y(j,k),$$

where $\alpha \in [0,1]$ is a balancing parameter. Note that RankClus sets $W_{XY}(i,j) = 0$ if an edge $\langle o_i, o_j \rangle \notin \mathbb{G}_i$; otherwise, $W_{XY}(i,j) = w_{o_i,o_j}$.

As shown in Definition 2, $\vec{r}_X$ and $\vec{r}_Y$ have mutual recursion formula, so RankClus needs to iterate the equations in Definition 2 until $\vec{r}_X$ and $\vec{r}_Y$ converge. As a result, Definition 2 incurs $O(t|\mathbb{E}|)$, where t is the number of iterations.

2.2.2. *Clustering part*

In this part, RankClus refines K clusters, *i.e.,* K partitions of X, used in the ranking part based on the rank score $\vec{r}_X$. RankClus first estimates posterior probability $\pi_{i,k}$ that represents a probability of target type node x_i belonging to cluster k by using the rank score $\vec{r}_X$. Then, RankClus constructs K-dimensional vector $s_{x_i} = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,K}\}$ for each $x_i \in X$, and it finally performs centroid-based clustering to update the K clusters. Here, we briefly review each procedure in the below.

Estimate the posterior probability $\pi_{i,k}$: RankClus estimates the posterior probability $\pi_{i.k}$ based on the *mixture generative model* [13]; $\pi_{i.k}$ is the probability that node x_i belonging to cluster k, where $k = 1, \ldots, K$.

Since $\pi_{i,k} = p(k|x_i) \propto p(x_i|k)p(k)$, RankClus here needs to compute $p(x_i|k)$ and $p(k)$. RankClus first regards $p(x_i|k)$ as the conditional rank of x_i in cluster k, which RankClus has already know in the ranking part. That is, RankClus obtains $p(x_i|k) = \vec{r_X}(i,k)$, where $\vec{r_X}(i,k)$ is a rank score for subgraph $\mathbb{G}_k$ from Definition 2. Then, it estimates $p(k)$, which represents the probability of an edge $\langle x, y \rangle$ ($x \in X$ and $y \in Y$) belongs to cluster k, by using EM-algorithm [14]. As a result, RankClus obtains the following posterior probability for each pair of x_i and cluster k.

Definition 3 (Posterior probability $\pi_{i,k}$) *Let $\pi_{i,k}$ be the posterior probability of node x_i belonging to cluster k, $\pi_{i,k}$ is defined as follows:*

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

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

Update K clusters: After estimating the posterior probability $\pi_{i,k}$, RankClus updates K clusters. First, it constructs K-dimensional feature vector $s_{x_i} = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,K}\}$ for each x_i. Then, RankClus determines K cluster centroids based on the following definition.

Definition 4 (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 follows:*

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

Next, RankClus evaluates each target type node $x_i \in X$ by using the following the distance measure:

Definition 5 (Distance) *Let $D(i,k)$ be the distance between node x_i and cluster X_k, $D(i,k)$ is defined as follows:*

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

Finally, RankClus updates K clusters by assigning $x_i \in X$ into a cluster X_k that shows the largest $D(i,k)$ value among K clusters.

As shown in Definition 3, estimating the posterior probability is not costful since we can get $\vec{r_X}(i,k)$ and $p(k)$ in $O(1)$. Also, updating K clusters requires $O(mK)$ times that is relatively smaller computational costs than the ranking part. Thus, the clustering part does not have a dominant cost in the RankClus algorithm.

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: Initialize
$t = 0$.
Generate initial K clusters $X_1^{(t)}, X_2^{(t)}, \ldots, X_K^{(t)}$.
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 $\vec{r}_Y(s, i)$ by Definition 2.
    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 3.
    end for
    Determine a centroid vector $\vec{s}_{X_i}$ by Definition 4.
  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 $D(j, k)$ by Definition 5.
    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.

2.2.3. *Algorithm*

We review the RankClus algorithm. The pseudo-code of RankClus is shown in Algorithm1. Algorithm 1 takes a bi-type information network $\mathbb{G}$ and a number of cluster K as inputs. Specifically, we can separate Algorithm 1 into the following steps:

- Step 0: Initialization.
  RankClus generates the initial K cluster of target type nodes in a random manner.
- Step 1: Ranking.
  RankClus computes rank scores of all nodes based on the current cluster structures.
- Step 2: Estimating the posterior probabilities.
  RankClus estimates the the posterior probabilities $\pi_{i,k}$ based on Definition 3 for all combinations of the target type nodes and the clusters.
- Step 3: Updating clusters.
  RankClus evaluates the distance from each target node x to each cluster centroid, and then it assigns the node to the nearest cluster.
- Repeat Step 1, 2 and 3 until all clusters converge.

2.3. *Computational cost of RankClus*

RankClus repeatedly performs the ranking process, the calculation of EM algorithm, and the clustering process on all the given objects, so the computational cost greatly increases as the number of data increases. As we described in Section , the time complexity for the ranking process is $O(t_1|\mathbb{E}|)$, where t_1 is the number of iterations of ranking, and $|\mathbb{E}|$ is the number of links. In clustering step, we need $O(K|\mathbb{E}| + K + mK)$ to estimate the mixture model, and $O(mK^2)$ to perform the cluster assignment process. Thus, the time complexity of the whole process is $O(t(t_1|\mathbb{E}| + t_2(K|\mathbb{E}| + K + mK) + mK^2))$, where t is the number of iterations of the whole algorithm, and t_2 is the iteration number of estimating the mixture model. This incurs high computational cost for large graphs.

3. PROPOSED METHOD

The goal of this paper is to reduce the computational cost of RankClus while keeping the clustering quality of the original RankClus algorithm [9]. To reduce the cost, we focus on the ranking part shown in Section 2.2.1 since it consumes a dominant part of the computation time. In this section, we first overview the central idea underlying our proposal and then give a full description of our proposed algorithm.

3.1. *Observation*

The basic idea of the proposed method is to reduce the computational cost for the ranking part. From Definition 2, the original RankClus algorithm needs to compute $\vec{r}_X(i,k)$ and $\vec{r}_Y(i,k)$. As discussed in Section 2.2.1, this computation incurs an expensive costs since the algorithm iterates this computation for all pairs of the nodes in $X \cup Y$ and the subgraphs until $\vec{r}_X(i,k)$ and $\vec{r}_Y(i,k)$ converge. Thus, it is essential to reduce the number of nodes in $X \cup Y$ whose rank score should be computed in each iteration.

To reduce the number of nodes to be computed, we have an important observation about the rank score that has not been stated and utilized in the existing works.

Observation 1 *In the ranking part, the rank score of each node does not significantly fluctuates. The rank score becomes almost converged value within early few iterations even though the original RankClus algorithm continues large number of iterations.*

For example, consider a bi-type information network dataset, which represents author-conference relationships taken from the DBLP bibliographic database. The dataset has 20 conferences as the target type nodes and 5,639 authors as the attribute type nodes. We set the number of clusters at $K = 4$ and performed the original RankClus algorithm on the dataset. Figure 3 shows how the rank scores fluctuate as the iteration proceeds on this dataset. By following Algorithm 1, the original RankClus algorithm performed five iterations until all rank scores converge; however, as we can see from Figure 3, most of the rank scores converged in the early iterations. That is, the original RankClus algorithm consumes unnecessary computational costs for the converged rank scores since it iteratively updates rank scores for all nodes regardless of whether the scores converged or not.

As we described in Section 2.2.2, the nodes with almost converged rank scores do not affect clustering results in the clustering part. This is because that the posterior probability used in the clustering part is composed of only the converged rank scores. Therefore, if the

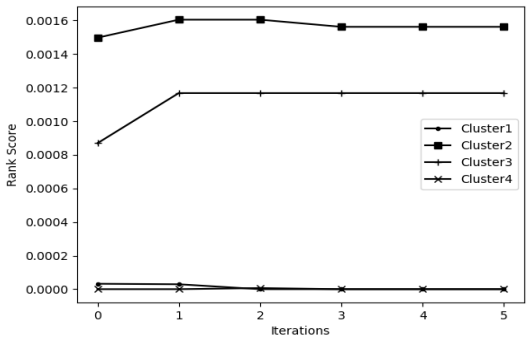

Fig. 3. Fluctuations of the rank scores in each iteration

nodes have almost converged rank scores, we can exclude nodes from the iterative rank score computations without significantly sacrificing the clustering quality.

3.2. *Change rate of rank score*

Based on the observation described in Section 3.1, our proposed method prunes unnecessary nodes whose rank scores almost converged from the ranking part. In order to specify the nodes with converged rank scores, we employ a new measure, named *change rate* R, that evaluates how the rank score increase/decrease compared to the previous iteration. The change rate shows how much the rank value has changed from the previous iteration. The formal definition is shown as follows:

Definition 6 (Change Rate) *Let $R^{(t)}(j,k)$ be the change rate of node x_j for subgraph $\mathbb{G}_k$ in the t-th iteration. The change rate $R^{(t)}(j,k)$ is defines by the following equation.*

$$R^{(t)}(j,k) = \frac{r^{(t)}(j,k)}{r^{(t-1)}(j,k)},$$

where $r^{(t)}(j,k)$ is $\vec{r_X}(j,k)$ in the t-th iteration if the node is a target type node; otherwise $r^{(t)}(j,k)$ is $\vec{r_Y}(j,k)$ in the t-th iteration.

As we can see from Definition 6, the change rate R evaluates the diference of the rank score between t-th and $(t-1)$-th iteration. Clearly, from the observation, the almost converged rank score does not increase/decrease its score compared with the score in the previous iteration. Hence, we can specify the nodes with almost converged rank scores by using the change rate R for each node.

To determine the almost converged rank score, we introduce two user-specified parameters, change-ratio parameter λ_R and stability parameter λ_I. The change-ratio parameter λ_R is a threshold of the change rate R; if a node has $R < \lambda_R$, our proposed method regards the rank score of the node almost converged. The stability parameter λ_I is a threshold to determine whether a node should be pruned or not; our proposed method prunes a node if it has almost converged rank scores (*i.e.,* $R < \lambda_R$) more than λ_I times. Formally, our proposed method prunes the following nodes.

Algorithm 2 Proposed method: Sequential Algorithm

Input: $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, Cluster Number K, λ_R, and λ_I
Output: $X_i (i = 1, 2, \cdots, K)$, $r_{\vec{X}}$, and $r_{\vec{Y}}$ for each X_i

// Step 0: Initialize
$t = 0$.
Generate initial K clusters $X_1^{(t)}, X_2^{(t)}, \ldots, X_K^{(t)}$.
Construct subgraphs $\mathbb{G}_i^{(t)}$ from $X_i^{(t)}$, and Y.
repeat
  // Step 1: Ranking for each cluster
  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 $\vec{r}_Y(s, i)$ by Definition 2.
      Compute $R^{(t)}(s, k)$ by Definition 6.
    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 3.
    end for
    Determine a centroid vector $\vec{s}_{X_i}$ by Definition 4.
  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 $D(j, k)$ by Definition 5.
    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
  // Step 4: Pruning
  for $i = 1$ to K **do**
    Construct subgraphs $\mathbb{G}_i^{(t+1)}$ from $X_i^{(t+1)}$ and Y.
    Obtain $\mathbb{P}_i$ by Definition 7.
    $\mathbb{V}_i^{(t+1)} = \mathbb{V}_i^{(t+1)} \backslash \mathbb{P}_i$.
  end for
  $t = t + 1$.
until No clusters are updated.

Definition 7 (Prunable Nodes) *Let $\mathbb{P}_k$ be a set of nodes that are pruned from $\mathbb{G}_k$, and $\bar{t}$ be the number of iterations when the ranking part terminated. The prunable nodes $\mathbb{P}_k$ is defined as follows:*

$$\mathbb{P}_k = \{u \in \mathbb{V}_k | \Theta(u, k) \geq \lambda_I\},$$

where $\Theta(u, k) = |\{R^{(t)}(u, k) | R^{(t)}(u, k) < \lambda_R \text{ for } t = 1, \ldots, \bar{t}\}|$; that is, $\Theta(u, k)$ is the number of $R^{(t)}(j, k)$ less than λ_R when t changes from 1 to $\bar{t}$.

In our proposed method, we prune the prunable nodes $\mathbb{P}_k$ for each subgraph $\mathbb{G}_k$ after the clustering part, so then it starts the next ranking part by using the reminder nodes.

3.3. *Algorithm*

Our proposed method reduces the number of nodes computed in the ranking part by using the change rate R and the prunable nodes $\mathbb{P}$ shown in Definition 6 and Definition 7, respectively.

The pseudo-code of our proposed method is shown in Algorithm 2.

Although our proposed method requires two additional user-specified parameters λ_R and λ_I, it is fundamentally based on the original RankClus algorithm described in Algorithm 1. However, different from the original algorithm, our proposed method additionally computes the change rate R in Step 1 (line 8), and based on the change rate R, we add one more step, named *Step 4* in lines 24-28. In Step 4, we identify the prunable nodes $\mathbb{P}_k$ for each subgraph $\mathbb{G}_k$ based on Definition 7 and the change rate R. After detecting the prunable node sets, our proposed method updates each subgraph $\mathbb{G}_k$ by removing $\mathbb{P}_k$ from $\mathbb{V}_k$. As a result, we reduce the size of each subgraph and performs the next ranking part on the subgraphs. As well as the original algorithm is shown in Algorithm 1, our proposed method terminates if the clustering result does not update.

3.4. *Multi-threading algorithm*

For further improving the computational efficiency of Algorithm 2, we here extend our algorithm so that it utilizes multi-threading techniques on a manycore CPU. Specifically, we employ thread-based parallelization into loop-blocks that compute ranks of nodes by pruning unnecessary nodes based on Definition 7. As shown in the previous section, our proposed algorithm needs to perform several computation steps for all clusters, *i.e.,* X_1^t, X_2^t, ..., X_K^t, except for (Step 3) in Algorithm 2. Thus, we apply task-wise parallelization for the steps to reduce the computation time of Algorithm 2. To balance task granuralities among threads, we divide the set of clusters $\{X_1^t, X_2^t, \ldots, X_K^t\}$ into equally sized T subsets in (Step 1), (Step 2), and (Step 4) in Algorithm 2, where T is a number of thread invoked in our parallel algorithm. After dividing the set of clusters, we finally assign each subset into a thread, and our algorithm computes the clusters in a task parallel manner.

Algorithm 3 shows the pseudocode of our multi-threading (parallel) algorithm, which is an extension of Algorithm 2 [12]. As we can see from Algorithm 3, the workflow is the same as that of Algorithm 2. Unlike our sequential algorithm in Algorithm 2, our parallel algorithm employs task-wise parallelization to compute multiple clusters simultaneously. Specifically, (line 5) in Algorithm 3, we assign each cluster into thread invoked in our algorithm since the ranking computation step is the most time-consuming part in RancClus algorithm as we discussed in Section 2.3. We also apply the task-wise parallelization into (Step 3) and (Step 4); that is, Algorithm 3 also performs our pruning technique described in Section 3.2 in the parallel computation manner. By dynamically removing prunable nodes, our parallel algorithm attempts to reduce computation times consumed in the succeeding iterations.

4. Experimental Analysis

We conducted extensive experiments to evaluate the effectiveness of our proposed methods. We designed our experiments to demonstrate that:

- **Efficiency:** Our proposed method shown in Algorithm 2 achieves faster computation time compared with the original RankClus algorithm; our proposal successfully avoids computing the rank scores of unnecessary nodes in the ranking part (Section 4.2).
- **Accuracy:** Although Algorithm 2 leads approximated clustering results, it does not sacrifice the clustering quality; our method outputs highly accurate clusters compared

Algorithm 3 Parallel proposed method

Input: $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, Cluster Number K, λ_R, and λ_I
Output: $X_i (i = 1, 2, \cdots, K)$, $r\vec{_X}$, and $r\vec{_Y}$ for each X_i

// Step 0: Initialize
$t = 0$.
Generate initial K clusters $X_1^{(t)}, X_2^{(t)}, \ldots, X_K^{(t)}$.
Construct subgraphs $\mathbb{G}_i^{(t)}$ from $X_i^{(t)}$, and Y.
repeat
  // Step 1: Ranking for each cluster
  for $i = 1$ to K **do in thread-parallel**
    for each $x_j \in X_i$ and $y_s \in Y$ **do**
      Compute rank score $\vec{r}_X(j, i)$ and $\vec{r}_Y(s, i)$ by Definition 2.
      Compute $R^{(t)}(s, k)$ by Definition 6.
    end for
  end for
  // Step 2: Get new attribute
  for $i = 1$ to K **do in thread-parallel**
    for each $x_j \in X_i$ **do**
      Estimate $\pi_{j,i}$ by Definition 3.
    end for
    Determine a centroid vector $\vec{s}_{X_i}$ by Definition 4.
  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 $D(j, k)$ by Definition 5.
    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
  // Step 4: Pruning
  for $i = 1$ to K **do in thread-parallel**
    Construct subgraphs $\mathbb{G}_i^{(t+1)}$ from $X_i^{(t+1)}$ and Y.
    Obtain $\mathbb{P}_i$ by Definition 7.
    $\mathbb{V}_i^{(t+1)} = \mathbb{V}_i^{(t+1)} \backslash \mathbb{P}_i$.
  end for
  $t = t + 1$.
until No clusters are updated.

with those of RankClus (Section 4.3).

- **Effectiveness:** The multi-threading algorithm introduced in Algorithm 3 effectively reduces the computation time of our proposed algorithm by increasing the number of threads; our parallel algorithm reduces the running time of Algorithm 2 under large T settings (Section 4.4).

- **Applicability:** Case-studies on real-world datasets shows that our algorithms output reasonable clustering results even though our proposal approximates the clustering results of RankClus. That is, our proposal can be a better alternative for the original algorithm (Section 4.5).

4.1. *Experimental Setup*

We compared the proposed method with the original RankClus algorithm [9]. All algorithms were implemented in C++11 (gcc 4.8.5) using `-O2` option, and we also used OpenMP for our parallel algorithm shown in Algorithm 3. All experiments were conducted on a CentOS server with an Intel (R) Xeon (R) E5-1620 3.50 GHz CPU and 128 GB RAM. Unless otherwise stated, we set the parameter α used in Definition 2 as 0.95, $\lambda_R = 0.8$, and $\lambda_I = 5$.

4.1.1. *Dataset*

We evaluated the algorithms on both synthetic and real-world datasets that are detailed in the below.

Synthetic dataset We evaluated the algorithms on five synthetic datasets that are generated in the existing work [9]. Each dataset has 45 target type nodes and 2,000 attributed type nodes, and the target type nodes can be divided into three clusters, *i.e.,* $K = 3$; the three clusters X_1, X_2, and X_3 have 10, 20, and 15 target type nodes, respectively. Based on the above settings, we generated five bi-type information networks by varying two parameter settings, P and T. Let Y be the set of attributed type nodes, $P = [P_1, P_2, P_3]$ represents the number of edges included in each subgraph; P_1, P_2, and P_3 are the number of edges in $X_1 \cup Y$, $X_2 \cup Y$, and $X_3 \cup Y$, respectively. $T = [T_{11}, T_{12}, T_{13}; T_{21}, T_{22}, T_{23}; T_{31}, T_{32}, T_{33}]$ denotes the probability that an edge is generated between two clusters; T_ij is the probability of linking from cluster i to j. The detailed setting of the five synthetic datasets, which are generated by [9], are listed as follows:

- **Dataset 1** (Medium separated & Medium density)

  $P = [1000, 1500, 2000]$,

  $T = [0.8, 0.05, 0.15; 0.1, 0.8, 0.1; 0.1, 0.05, 0.85]$

- **Dataset 2** (Medium separated & Low density)

  $P = [800, 1300, 1200]$,

  $T = [0.8, 0.05, 0.15; 0.1, 0.8, 0.1; 0.1, 0.05, 0.85]$

- **Dataset 3** (Medium separated & High density)

  $P = [2000, 3000, 4000]$,

  $T = [0.8, 0.05, 0.15; 0.1, 0.8, 0.1; 0.1, 0.05, 0.85]$

- **Dataset 4** (Highly separated & Medium density)

  $P = [1000, 1500, 2000]$,

  $T = [0.9, 0.05, 0.05; 0.05, 0.9, 0.05; 0.1, 0.05, 0.85]$

- **Dataset 5** (Highly separated & Low density)

  $P = [800, 1300, 1200]$,

  $T = [0.9, 0.05, 0.05; 0.05, 0.9, 0.05; 0.1, 0.05, 0.85]$

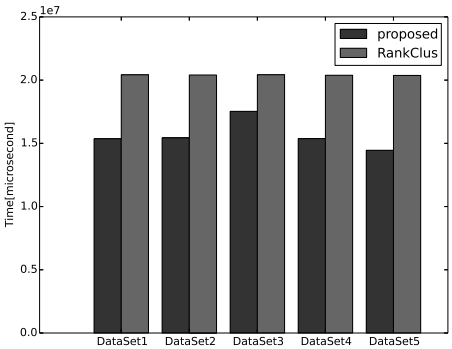

Fig. 4. Runtimes on synthetic datasets.

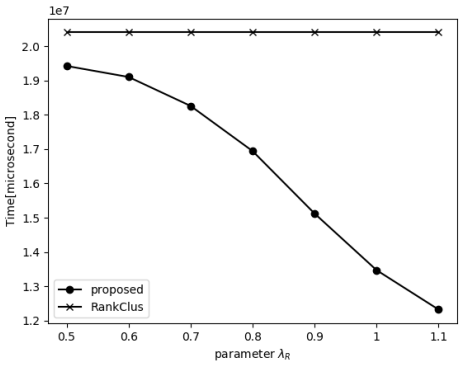

Fig. 5. Runtimes by varying λ_R.

Real-world dataset In addition to the synthetic datasets, we also used a real-world dataset for evaluating the effectiveness of our proposed algorithms. In this paper, we constructed an author-conference bi-type information network named *DBLP dataset* that is extracted from the DBLP bibliographic database [b]. We collected publication lists of 20 representative conferences from DBLP, and we extracted 5,639 authors who have been published at least two papers on the 20 conferences. If an author has published papers on a conference, we link the author and the conference by an edge with a weight value, which represents many papers the author published at the conference. Also, if two authors have been co-authors, we connect the two authors by an edge; we set many papers coauthored with the two authors as a weight value for each edge. We did not set any edges among the 20 conferences.

4.2. *Efficiency*

We compared the running time of the proposed method compared with the original RankClus algorithm by using the five synthetic datasets described in Section 4.1.1. Figure 4 shows the results of each algorithm on the synthetic datasets. As we can see from the results, our proposed method successfully reduces more than 30% of the computation time consumes by the original RankClus algorithm.

We also evaluated the effect of the user-specified parameters, λ_R and λ_I. In this evaluation, we fixed one of the two parameters and varied the other one on Dataset 3. Figure 5 shows the runtimes when we varied λ_R from 0.5 to 1.1 with fixed λ_I value (*i.e.,* $\lambda_I = 5$). As shown in Figure 5, the computation time gradually decreased as the size of λ_I increases. This is because that, as shown in Figure 3, most nodes typically decrease their rank scores as iteration proceeds. Thus, by setting λ_R values around 1, our proposed method can prune such nodes in the early iterations. As well as λ_R, we also evaluated the effect of λ_I values on Dataset 3. Figure 6 shows the runtimes when we vaired λ_I from 2 to 8 with fixed λ_R value (*i.e.,* $\lambda_R = 0.8$). As shown in Figure 6, the runtime gradually increased as the size of λ_I becomes large. Clearly, large λ_I values require more strict pruning conditions than small λ_I values, hence the size of prunable nodes $|\mathbb{P}_k|$ becomes small as λ_I increases.

[b] `https://dblp.uni-trier.de/`

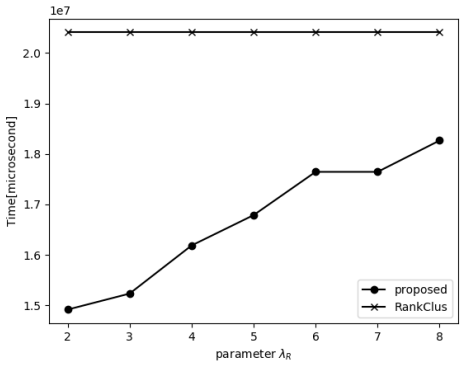

Fig. 6. Runtimes by varying λ_I.

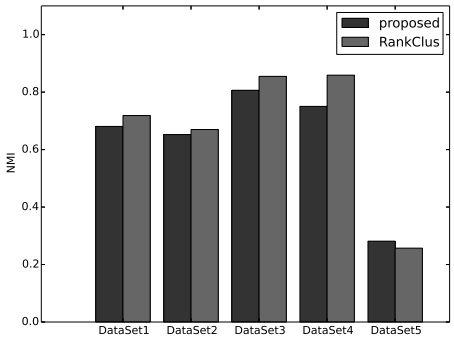

Fig. 7. NMI on synthetic datasets.

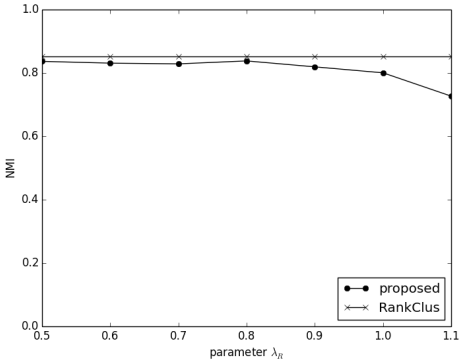

Fig. 8. NMI by varying λ_R

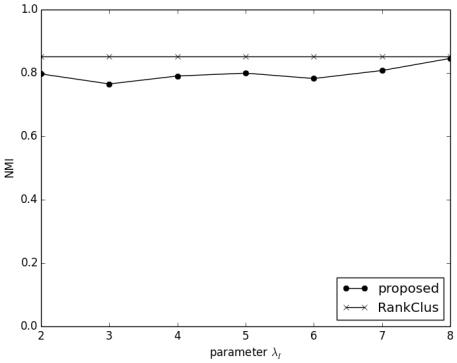

Fig. 9. NMI by varying λ_I.

The results shown in Figure 4, 5, and 6 confirm that the superiority to the original RankClus algorithm in terms of runtimes.

4.3. *Accuracy*

We experimentally confirm the accuracy of the clustering results produced by the proposed method. In order to evaluate the accuracy of clustering results, we employed *NMI* (Normalized Mutual Information) [15] that is an information-theoric measurement. NMI measures clustering accuracy by comparing the ground truth and a clustering result. NMI takes a value between 0 and 1; NMI returns 1 if the two clusters are completely same while it outputs 0 if the two clusters are different. In this evaluation, we compared the ground truth clusters of each synthetic dataset and their corresponding clustering results produced by our proposal and the original RankClus algorithm. Figure 7 shows the results of the five synthetic datasets. As we demonstrated in Figure 7, our proposed method shows very similar NMI values compared with the original RankClus algorithm even though our proposal dynamically prunes nodes from the ranking part. That is, our proposed method does not significantly degrade

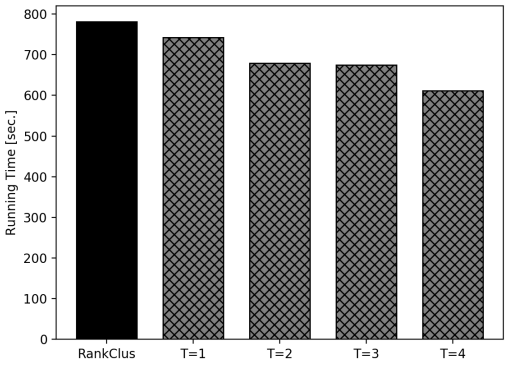

Fig. 10. Effects of the number of thread in Algorithm 3 on DBLP-dataset

the clustering accuracy compared with those of the original algorithm.

As well as Section 4.2, we also evaluated the effect of the parameters, λ_R and λ_I. Figure 8 and Figure 9 show how NMI value changes by varying λ_R and λ_I, respectively. In Figure 8, we fixed $\lambda_I = 5$, and varied λ_R from 0.5 to 1.1. Similarly, in Figure 9, we fixed $\lambda_R = 0.8$, and varied λ_I from 2 to 8. As we can see from Figure 8, our proposed method keeps high accuracy (*i.e.,* NMI ≥ 0.8) for all parameter settings except for $\lambda_R = 1.1$. By using the change rate measure shown in Definition 6, our proposed method prunes only the nodes whose rank score have already converged. Thus, our pruning technique does not significantly affect the clustering quality. In contrast, as shown in Figure 9, the NMI values gradually increase by setting larger λ_I values. This is because that the nodes can be pruned before their rank scores enough converge if we set small λ_I values since the rank scores drastically fluctuate in the early iterations. Hence, it is reasonable to set relatively large values for λ_I.

From Figure 7, 8, and 9, we confirmed that our algorithm shows good accuracy compared with the original algorithm even though our proposal employs node pruning techniques.

4.4. *Effectiveness of our multi-threading approach*

We conduct an experimental analysis to evaluate the effectiveness of our multi-threading technique shown in Algorithm 3. In this evaluation, we compared the running time of RankClus with that of our parallel algorithm in Algorithm 3 on DBLP dataset. For our proposed algorithm, we varied the number of threads T from 1 to 4 to test the effectiveness of thread-based parallelization. Note that the parallel algorithm is equivalent to Algorithm 2 if we set $T = 1$.

Figure 10 shows the running time comparisons between RankClus and our algorithm. In Figure 10, $T = 1$, $T = 2$, $T = 3$, and $T = 4$ in x-axis indicate our parallel algorithm using 1 thread, 2 threads, 3 threads, and 4 threads, respectively. In the case of $T = 1$, our proposed method uses only the node pruning technique described in Section 3.2. As shown we can see in Figure 10, our proposed method shows faster clustering time compared with RankClus as well as the results in Section 4.2. Furthermore, we can observe from the figure that our parallel algorithm successfully reduces the running time by increasing the number of threads. These results indicate that our parallelization approach is effective in reducing the running

Table 6. Clustering and ranking results on the DBLP dataset.

Rank	Cluster 1	Cluster 2	Cluster 3	Cluster 4
1	SIGMOD	KDD	SIGIR	NIPS
2	VLDB	ICDM	TREC	ICML
3	EDBT	SDM	ECIR	UAI
4	PODS	PAKDD	JCDL	COLT
5	ICDT	PKDD	ECDL	
6		ECML		

time of RankClus.

4.5. *Case-study on real-world DBLP dataset*

To demonstrate the impact of our algorithm on real-world applications, we give a case study of its clustering performance on DBLP dataset. In this case study, we regard the 20 conferences as the target type nodes, and we performed our proposed method on the real-world dataset by setting the number of clusters as $K = 4$.

Table 6 shows the clusters and the ranks of the 20 conferences extracted by our proposed algorithm. As we can see from the table, our proposed method detected reasonable clustering and ranking results from the real-world dataset even though our algorithm produces approximated results as we stated in Section 4.3. Specifically, Cluster 1, 2, 3, and 4 in Table 6 represent the groups of database, data mining, information retrieval, and machine learning, respectively. For a particular interest, our proposed method groups ECML into Cluster 2 (data mining) instead of Cluster 4 (machine learning) even though ECML stands for Europian Conference on *Machine Learning*. The reason is apparent; ECML has been collocated with PKDD since 2008, and thus the target scopes and authors of ECML and PKDD significantly overlapped. Hence, our algorithm regards ECML as a member of the data mining cluster. These results imply that our algorithm can be another option for the research community for clustering large heterogeneous graphs.

5. Related Work

Since ranking and clustering are the fundamental graph mining tools, various approaches [16] have been studied for some decades. Here, we review some of the successful algorithms from the ranking and clustering perspectives.

5.1. *Ranking algorithm*

The ranking is a graph mining method to evaluate the importance of each node included in a graph; PageRank [2] and HITS [3] are the most representative ranking algorithms that employ random-walk based graph analysis. The main idea underlying PageRank is that a node that has many edges connected with important nodes should be more important. On the other hand, HITS focuses on the two types of nodes, *authority* and *hub*: the authority is a node that propagates importance to its neighbor nodes, while the hub is a node that mediates between authority nodes. Based on the above ideas, PageRank and HITS estimate the importance of all nodes by iteratively performing random-walks on all nodes included in

the graph. Although they are successful in a wider range of Web-based applications, PageRank and HITS have two limitations. First, they can not handle heterogeneous graphs; it is thus difficult to apply PageRank and HITS to the bi-type information networks shown in Section 2. Second, they do not consider the clusters for estimating the importance; hence, PageRank and HITS output ranking results without splitting nodes into their corresponding clusters.

In order to perform the ranking on heterogeneous graphs, ObjectRank [17, 4] has been proposed. However, as well as PageRank and HITS, this method does not capture the cluster structures included in the graph, and so, it fails to compute cluster-aware ranking results. Similarly, to adequately capture the cluster structures for the ranking, Manifold Ranking [18] is one of the promising ways for homogeneous graphs. Different from our proposed method, Manifold Ranking, however, can not compute heterogeneous graphs; it is thus difficult to apply it to the bi-type information networks.

5.2. *Clustering algorithm*

The problem of finding clusters in a graph has been studied for some decades. Graph partitioning algorithms [19, 20] are natural choices for this problem. Since cluster structures are highly complex, several clustering algorithms have been recently introduced. Here we review some of the more successful methods.

Modularity-based algorithm [21, 5, 6] is one of the most popular algorithms for finding clusters included in graphs. The main idea of modularity-based algorithms is to find groups of nodes that have a lot of inner-group edges and few inter-group edges. Since they can compute large graphs efficiently, they are used in various applications such as Web analysis, biological data analysis, network security and so on. However, despite the efficiency of the algorithms, it is recently pointed out that these methods fail to fully reproduce the ground-truth [22]. To overcome the above limitation, structural graph clustering algorithms [23, 24] have recently proposed. The structural graph clustering is an extension of the traditional density-based clustering method DBSCAN [25]. Thus the structural graph clustering can successfully find clusters as well as hubs and outliers in a graph, and as a result, it typically produces better clustering results than those of the modularity-based algorithms.

Although the above approaches are used to understand the cluster structures in a graph, their applications are limited in homogeneous graphs; that is, they can not handle multiple types of nodes like a bi-type information network in their clustering procedure. In contrast, RankClus [9], proposed by Sun *et al.,* is the state-of-the-art ranking-based clustering algorithm on heterogeneous graphs. However, as we described in Section 2, RankClus requires high computational costs. In this paper, we proposed an efficient algorithm for RankClus by providing an efficient node pruning technique shown in Section 3. As a result, as we experimentally confirmed in Section 4, our proposed method achieved faster clustering while keeping the clustering quality of RankClus.

6. Conclusion

In this paper, we proposed two algorithms to improve the efficiency of RankClus algorithm for large heterogeneous graphs. The first one is the node-pruning algorithm that reduces unnecessary nodes computed in the ranking part of RankClus to avoid iterative computations for all nodes and edges. In order to find the prunable nodes, we employed an evaluation

criterion named change rate, and our algorithm drops nodes whose change rate score becomes stable during the iterative computations. The second algorithm is a multi-threading method that is an extension of our node-pruning algorithm. In the node-pruning algorithm, we need to perform the ranking and the clustering for all clusters iteratively. Thus, we employed task-wise parallelization to speed up the node-pruning in our parallel algorithm. Our extensive evaluations using both synthetic and real-world datasets showed that our proposed node-pruning method is faster than RankClus 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 node-pruning method by increasing the number of thread invoked in the algorithm. RankClus now plays an essential role in current and prospective Web-based systems and applications in various disciplines. By providing our efficient algorithms, it will help to improve the effectiveness of future applications.

Acknowledgements

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. Makoto Onizuka, Toshimasa Fujimori, and Hiroaki Shiokawa. Graph Partitioning for Distributed Graph Processing. *Data Science and Engineering*, 2(1):94–105, Mar 2017.
2. Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst.*, pages 107–117, 1998.
3. Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, pages 604–632, 1999.
4. Tomoki Sato, Hiroaki Shiokawa, Yuto Yamaguchi, and Hiroyuki Kitagawa. FORank: Fast ObjectRank for Large Heterogeneous Graphs. In *Companion Proceedings of the The Web Conference 2018*, pages 103–104, 2018.
5. V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E.L.J.S. Mech. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Yasutoshi Ida, and Machiko Toyoda. Adaptive Message Update for Fast Affinity Propagation. In *Proc. KDD*, pages 309–318, 2015.

12. 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, New York, NY, USA, 2018. ACM.
13. Andrew Y. Ng and Michael I. Jordan. On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.
14. Jeff Bilmes. A gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical report, 1998.
15. Alexander Strehl and Joydeep Ghosh. Cluster Ensembles — a Knowledge Reuse Framework for Combining Multiple Partitions. *J. Mach. Learn. Res.*, pages 583–617, 2003.
16. Yasuhiro Fujiwara, Yasutoshi Ida, Hiroaki Shiokawa, and Sotetsu Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 1561–1567, 2016.
17. Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *Proc. VLDB*, pages 564–575, 2004.
18. Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Saliency Detection via Graph-Based Manifold Ranking. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3166–3173, Washington, DC, USA, 2013.
19. Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 888–905, 2000.
20. Ulrike Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, pages 395–416, 2007.
21. M. E. J. Newman and M. Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review*, E 69(026113), 2004.
22. Santo Fortunato and M Barthelemy. Resolution Limit in Community Detection. *Proceedings of the National Academy of Sciences*, Jan 2007.
23. Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas 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)*, pages 824–833, New York, NY, USA, 2007. ACM.
24. 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.
25. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.