Efficient Distributed Clustering Algorithms on Star-Schema Heterogeneous Graphs

Chen, Lu; Gao, Yunjun; Huang, Xingrui; Jensen, Christian S.; Zheng, Bolong

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2020.3047631

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

Efficient Distributed Clustering Algorithms on Star-Schema Heterogeneous Graphs

Lu Chen, Yunjun Gao, Member, IEEE, Xingrui Huang, Christian S. Jensen, Fellow, IEEE, Bolong Zheng

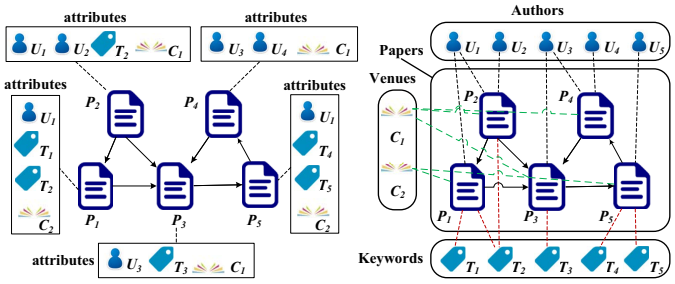
Abstract—Many datasets including social media data and bibliographic data can be modeled as graphs. Clustering such graphs is able to provide useful insights into the structure of the data. To improve the quality of clustering, node attributes can be taken into account, resulting in attributed graphs. Existing attributed graph clustering methods generally consider attribute similarity and structural similarity separately. In this paper, we represent attributed graphs as star-schema heterogeneous graphs, where attributes are modeled as different types of graph nodes. This enables the use of personalized pagerank (PPR) as a unified distance measure that captures both structural and attribute similarities. We employ DBSCAN for clustering, and we update edge weights iteratively to balance the importance of different attributes. The rapidly growing volume of data nowadays challenges traditional clustering algorithms, and thus, a distributed method is required. Hence, we adopt a popular distributed graph computing system Blogel, based on which, we develop four exact and approximate approaches that enable efficient PPR score computation when edge weights are updated. To improve the effectiveness of the clustering, we propose a simple yet effective edge weight update strategy based on entropy. In addition, we present a game theory based method that enables trading efficiency for result quality. Extensive experiments on real-life datasets offer insights into the effectiveness and efficiency of our proposals.

Index Terms—Heterogeneous graph, Clustering, Distributed Processing, Algorithm

1 Introduction

Graphs are used to model the relationships between objects in many settings such as web, social networks, and biological networks. Clustering is able to offer useful insights into the characteristics of a graph. The goal of clustering is to partition a set of data objects into a set of clusters, such that objects in a cluster are similar to each other, while objects in different clusters are dissimilar. Recently, additional information (e.g., text) has been taken into account to help improve the quality of clustering, resulting in so-called attributed graphs. Each node in an attributed graph may have many different kinds of attributes. DBLP is a typical bibliographical database that can be modeled as an attributed graph. Specifically, consider Fig. 1(a), where papers are modeled as nodes, citations are modeled as edges, and each paper has several attributes (e.g., keywords, authors, and venues).

When clustering an attributed graph, it is important to consider both structural and attribute similarities. In particular, clustering based on the features of objects (i.e., attribute similarity) has been investigated widely, and many methods (e.g., k-means [1], DBSCAN [2], etc. [3]) have been proposed in the literature. In addition, clustering based on topological structures has also been studied, yielding methods that utilize normalized cuts [4], modularity [5], structural density [6], or flows [7] for clustering. Most of the existing clustering algorithms for attributed graphs consider attribute similarity and structural similarity separately [8], [9], [10], [11], [12], where separate distance measures are used to measure attribute similarity and structural similarity. In addition, they ignore differentiating the importance of different attributes.



(a) Attributed graph (b) Star-schema heterogeneous graph

Fig. 1. Graph Models of Bibliographic Information

Inspired by a recent line of studies [13], [14], we transform attributed graphs to star-schema heterogeneous graphs, where attributes (e.g., keywords, authors, and venues) are modeled as different types of graph nodes that are connected to the hub nodes (e.g., papers), as illustrated in Fig. 1(b). By doing this, we are able to utilize Personalized PageRank (PPR) to achieve a unified notion of similarity, and model the importance of different attributes using the edge weight that is updated iteratively according to its contribution to the clustering result. We give two examples below.

Example 1. Bibliographic information is often modeled as an attributed graph, where papers are connected by citations, and each paper has several attributes (e.g., keywords, authors, and venues). An attributed graph can in turn be represented as a star-schema heterogeneous graph as depicted in Fig. 1(b), which contains four types of nodes. Each paper, modeled as a node, has edges of different types that connect it to other papers and to descriptive attribute values of different types, where the attribute values are also modeled as nodes. The application of clustering to a graph model of bibliographic information can help detect communities.

- *L. Chen, Y. Gao (Corresponding Author), and X. Huang are with the College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China. E-mail: {luchen, gaoyj, xrhuang}@zju.edu.cn.*
- *C. S. Jensen is with the Department of Computer Science, Aalborg University, Aalborg, Denmark. E-mail: csj@cs.aau.dk.*
- *B. Zheng is with the School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail: bolongzheng@hust.edu.cn.*

Example 2. Social image data can also be modeled as an attributed graph. Images are connected to their k most similar images [15], [16] according to their contents, and each image has attributes (e.g., keywords and users). Again, the attributed graph model can be transformed to a star-schema heterogeneous graph model. In this example, the model contains three types of nodes. The application of clustering to a social image graph can help generate image recommendations.

As discussed in two examples above, clustering of attributed graphs is useful in many application domains, such as recommendation systems, community detection, and so on. The last years featured a rapid growth of data volume. Thus, the attributed graphs cannot be stored or processed in one centralized machine any more, and a distributed method is required. Motivated by these, we explore distributed clustering of attributed graphs. Although distributed graph clustering methods have been studied [17], [18] recently, they are not designed for attributed graphs that consider both structure and attribute similarities. To our best knowledge, this is the first attempt for efficient and effective distributed clustering of attributed graphs. To support efficient and effective attributed graph clustering, three challenges must be addressed.

The first challenge is *how to define a unified distance measure* that captures both structural and attribute similarities. Existing studies [13], [14] transform attributed graphs into star-schema heterogeneous graphs, and use *fixed-length* random walks to define a unified distance measure. Extending this line of work, we consider *arbitrary-length* random walks and ensure the symmetric property required by clustering methods. Specifically, we define a symmetric distance measure based on PPR that takes mutual neighborhoods into consideration.

The second challenge is *how to increase the scalability*, as the data volume increases rapidly nowadays. To address this, we adopt a popular distributed graph processing system Blogel that supports scalable data storage and data processing over multiple work nodes instead of a single work node.

The third challenge is *how to further improve the clustering performance*. We present a simple yet efficient weight update strategy based on entropy of intermediate clustering results to control the importance of different attributes, and we propose distributed PPR score computation methods over Blogel. We also present two types of optimizations to boost the efficiency. The first one is to avoid recomputing PPR scores when edge weights are updated: (i) we partially compute and store the PPR scores, and then recompute the exact scores based on the stored values; and (ii) we perform approximate updates of PPR scores to simulate edge weight updates. The second one is to reduce the communication cost: particularly, we sample the message transmission during distributed PPR score computation, which trades accuracy for efficiency. In addition, a game-theory based clustering approach is developed to refine the result by trading efficiency for effectiveness.

To sum up, the key contributions are summarized as follows:

- We present an iterative framework for distributed clustering of attributed graphs, which utilizes PPR on the transformed heterogeneous graphs to define a unified distance and uses DBSCAN for clustering.
- We propose four distributed PPR score computation methods over Blogel to avoid PPR score re-computations and reduce the communication cost in the distributed environment, thus further improving the efficiency of clustering.
- We develop a simple yet efficient strategy to update the weights of different attribute types iteratively, and propose a distributed game theory based approach that refines clustering results, to improve the effectiveness of clustering.
- We report on extensive experiments using two real-life datasets that offer insights into the scalability, efficiency, and effectiveness of our proposed methods.

A preliminary report of this work is published in [19]. We extend mainly it in this paper, by (1) studying *distributed* clustering on star-schema heterogeneous graphs; (2) including related work on distributed clustering over graphs; and (3) conducting experimental evaluation that investigates the efficiency, effectiveness, and scalablilty of the corresponding proposed methods. Note that, due to space limitation, we remove the incremental methods in previous conference version that are designed for *centralized* clustering on star-schema heterogeneous graphs.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents preliminaries. Section 4 describes our clustering framework. Sections 5 and 6 detail distributed PPR score computation methods and the game theory based method, respectively. Experimental results and our findings are reported in Section 7. Finally, Section 8 concludes and offers directions for future work.

2 Related Work

In this section, we provide a brief overview of previous work on clustering over heterogeneous graphs and distributed clustering over graphs, respectively.

2.1 Clustering over Heterogeneous Graphs

Attributed graphs can be modeled as star-schema heterogeneous graphs [13], [14], [20]. Two surveys on the clustering of heterogeneous graphs exist [9], [20]. Unlike homogeneous graphs, attribute information is integrated into the clustering analysis on heterogeneous graphs. Sun et al. [10] propose a model-based clustering algorithm that takes into account incomplete attribute information and the graph structure. Qi et al. [8] develop a clustering algorithm based on heterogeneous random fields to model the structure and content of social network graphs with outlier links. Cruz et al. [21] integrate structural and compositional dimension for the purpose of community detection. A recent density-based clustering model [22] aims to detect clusters by considering both the graph connections and the node attributes. More recently, Li et al. [11] use a semi-supervised method, and Baroni et al. [12] provide an efficient method, for large attributed graphs. Nonetheless, these studies consider the attribute similarity and the structural similarity separately. To consider attribute and structural similarities together, one line of study [13], [14] uses a fixed-length random walks as a unified distance measure with limited scalability. In contrast, we utilize Personalized PageRank, thus taking random walks of any length into consideration. We aim to design more efficient and scalable star-schema heterogeneous graph clustering methods.

Instead of using a distance measure, a Bayesian probabilistic model is developed for attributed graphs [23]. In addition, user information [24], [25], [26] and social influence [27] can also be integrated into clustering analysis to improve clustering quality. Furthermore, clustering can also be integrated with other mining tasks (e.g., ranking [28], [29], [30], [31]) to improve the overall

TABLE 1
Symbols and Description

Notation	Description
$G(V, E, \varphi, \psi, w)$	heterogeneous graph with a set V of nodes, a set E of edges, a node mapping function φ, a edge mapping function ψ, and a weight function w
$\mathcal{A}$, $\mathcal{R}$	sets of node and edge types
$\|\mathcal{A}\|$, $\|\mathcal{R}\|$	numbers of node and edge types
$\mathcal{A}_i$, $\mathcal{R}_i$	particular node and edge type
$\|\mathcal{A}_i\|$	number of objects of type $\mathcal{A}_i$
u, v	nodes in a heterogeneous graph
e	edge in a heterogeneous graph
c_i, c	weight constant for $\mathcal{A}_i$ and for all attribute types
$\mathbf{P}$	transition probability matrix
$p[v, u]$	transition probability from node v to node u
$h(v, u)$	incidence function: $h(v, u) = 1$ if $(v, u) \in E$
α	positive restart probability of a random walk
r_{max}	a threshold (i.e., an error bound) for PPR
$\pi(s, t)$	PPR score from start node s to target node t
$\hat{\pi}(s, t)$	reserve of $\pi(s, t)$ in backward search method
$r(s, t)$	residue of $\pi(s, t)$ in backward search method
$\hat{\Pi}_v$	reserve map contains pairs $(u, \hat{\pi}(u, v))$ $(u \in V)$
R_v	residue map contains pairs $(u, r(u, v))$ $(u \in V)$
$d(s, t)$	symmetric distance between nodes s and t
δ	positive real value used to define density
$minPts$	positive integer used to define density
β	real value in the range [0, 1]
r_s	sample rate in the range (0, 1]
n_w	number of work nodes

performance through joint processing. In this paper, we do not assume the availability of additional information, and do not assume the integration with other mining tasks. Hence, we solve a different and less specialized problem.

2.2 Distributed Clustering over Graphs

Although distributed graph clustering methods have been studied [32], [33], there is, to the best of our knowledge, no previous work that investigates distributed clustering over attributed graphs. Several studies [34], [35] explore distributed clustering on heterogeneous sensor networks. However, those heterogeneous clustering methods aim to save energy, and only focus on the specific heterogeneous graphs, which is different from our problem studied in this paper. In contrast, we aim at distributed clustering on attributed graphs modeled as heterogeneous graphs, which aims to improve its scalablity, efficiency, and effectiveness. McClean et al. [36] study clustering of heterogeneous distributed databases, and propose a hybrid metric to measure the distance between two databases. However, the problem is different from our work, as we focus on graphs and present a distributed method to cluster one graph dataset, while they develop a centralized method to cluster multiple heterogeneous datasets.

Many distributed graph processing platforms exist, which includes MapReduce [37], Pregel [38], Giraph++ [39], Blogel [40], GraphLab [41], Trinity [42], Spark [43], and GraphX [44]. We choose Blogel as the underlying distributed framework, because i) it is a popular Pregel-like system, which is good for iterative graph query processing and supports massage combiners; ii) it extends Pregel with a block-computing functionality, which can achieve high performance in processing large real world graphs; and iii) it provides global interfaces for programming algorithms in C++.

3 Preliminaries

We first introduce attributed and heterogeneous graphs, and then, we describe Personalized PageRank as a unified distance measure. Table 1 summarizes frequently used notations in this paper.

3.1 Attributed Graphs

Graphs are used to model real-world settings consisting of objects and relationships or interactions among the objects. For attributed graphs, each node in the graph can also be associated with various attribute values to represent valuable information.

Definition 1. **(Attributed Graph)** A attributed graph is defined as a directed graph $G = (V, E)$, in which V denotes a set of nodes and E represents a set of edges. Here, each node v in V is associated with a set of attributes.

Fig. 1(a) gives an attributed graph example, where $V = \{P_1, P_2, P_3, P_4, P_5\}$. Each paper has a set of attributes. For instance, P_2 has authors U_1 and U_2, a keyword T_2, and the venue C_1. Next, we present the definition of clustering in a attributed graph.

Definition 2. **(Clustering in a Attributed Graph)** Clustering in a attributed graph G is to partition the node set V in G into k disjoint sets V_i $(1 \leq i \leq k)$, where $V = \cup_{i=1}^{k} V_i$ and $V_i \cap V_j = \emptyset$ for any $i \neq j$. It needs to achieve a good balance between the following two objectives: (i) nodes in a cluster are close to each other in terms of structure, while nodes in different clusters are distant from each other; and (ii) nodes in a cluster have similar attributes, while nodes in different clusters have quite different attributes.

The first objective concerns structural similarity, whereas the second concerns attribute similarity. In Fig. 1(a), the papers can be clustered into two sets $\{P_1, P_2\}$ and $\{P_3, P_4, P_5\}$.

3.2 Heterogeneous Graphs

Since heterogeneous graphs can model objects and relationships of different types, we transform attributed graphs to heterogeneous graphs for a unified representation.

Definition 3. **(Heterogeneous Graph)** A heterogeneous graph is defined as a directed and weighted graph $G = (V, E, \varphi, \psi, w)$ with (i) a node type mapping function $\varphi : V \rightarrow \mathcal{A}$, that maps each node v $(\in V)$ to a node type $\mathcal{A}_i$ $(\in \mathcal{A})$; (ii) an edge type mapping function $\psi : E \rightarrow \mathcal{R}$, that maps each edge e $(\in E)$ to an edge type $\mathcal{R}_i$ $(\in \mathcal{R})$; and (iii) a weight function $w : E \rightarrow \mathsf{R}^{\star}$, that maps each edge e $(\in E)$ to a non-negative real value. Here, $|\mathcal{A}| > 1$ and $|\mathcal{R}| > 1$.

We use $w(e)$ to represent the weight of edge e, and we use $v \in \mathcal{A}_i$ to denote that node v has the node type $\mathcal{A}_i$.

Different categories of heterogeneous graphs exist, including star-schema, bipartite, and multiple Hub [9]. In this paper, we only focus on the star-schema graphs, since they are quite versatile and can model attributed graphs. A star-schema graph contains a main node type and several attribute node types. Nodes of the former type are called hub nodes, and nodes of the latter types are called attribute nodes.

Fig. 1(b) depicts a star-schema heterogeneous graph for bibliographic information. Let $\mathcal{A}_0$ be a hub node type, and $\mathcal{A}_i$ $(1 \leq i \leq |\mathcal{A}| - 1)$ be attribute node types. As illustrated in Fig. 1(b), papers P_1, ..., P_5 are hub nodes of type $\mathcal{A}_0$; while authors U_1, ..., U_5, keywords T_1, ..., T_5, and publication venues C_1, ..., C_2 are attribute nodes of types $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, respectively.

In addition, edges of type $\mathcal{R}_0$ connects two hub nodes, while $\mathcal{R}_i$ $(1 \leq i \leq |\mathcal{R}| - 1)$ connect a hub node and an attribute node of type $\mathcal{A}_i$. Back to Fig. 1(b), the black solid lines denote the edges of type $\mathcal{R}_0$ that connect two papers, while the black, red, and green dotted lines represent the edges of types $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$, respectively, which connect a paper and an attribute node (i.e., author, keyword, or venue).

A heterogeneous graph G can be encoded in a $|V| \times |V|$ incidence matrix $\mathbf{H}$ where

$$h(v,u) = \begin{cases} 1 & (v,u) \in E \\ 0 & (v,u) \notin E \end{cases} \tag{1}$$

We assume that, for each hub node v, the sum of the weights of the edges from v to all outgoing nodes $u \in \mathcal{A}_i$ equals to a constant c_i. The constant c_i is used to capture the importance of the node type $\mathcal{A}_i$ when forming clusters. For simplicity, c_i is evenly distributed among the outgoing edges that have type $\mathcal{A}_i$. We fix $c_0 = 1$ and $\sum_{i=1}^{|\mathcal{A}|-1} c_i = c$, where c is a constant, so that c_i can be updated according to its contribution to the clustering result. In our experiments, c is set to the number of attribute types as default, and c_i $(1 \leq i \leq |\mathcal{A}| - 1)$ for each attribute type is initialized as 1. In addition, we also explore the impact of c on the clustering via experiments in Section 7. As a result, the weight of edge $e = (v,u)$ is defined as follows.

$$w(v,u) = \begin{cases} \frac{c_i \cdot h(v,u)}{\sum_{x \in \mathcal{A}_i} h(v,x)} & v \in \mathcal{A}_0 \wedge u \in \mathcal{A}_i \\ \frac{h(v,u)}{\sum_{x \in V} h(v,x)} & v \notin \mathcal{A}_0 \end{cases} \tag{2}$$

In Fig. 1(b), let c_i $(0 \leq i \leq 3)$ be initialized to 1. For hub node P_2, we can get that $w(P_2,P_1) = 0.5$, $w(P_2,P_3) = 0.5$, $w(P_2,U_1) = 0.5$, $w(P_2,U_2) = 0.5$, $w(P_2,T_2) = 1$, and $w(P_2,C_1) = 1$. For attribute node U_1, we can get that $w(U_1,P_1) = 0.5$ and $w(U_1,P_2) = 0.5$. Next, we formalize clustering on a star-schema heterogeneous graph.

Definition 4. **(Clustering on a Star-Schema Heterogeneous Graph)** Clustering on a star-schema heterogeneous graph G is to partition nodes V in G into k disjoint sets V_i $(1 \leq i \leq k)$, where $V = \cup_{i=1}^{k} V_i$ and $V_i \cap V_j = \emptyset$ for any $i \neq j$, to achieve the goal that nodes within one cluster are close to each other in terms of structure and attribute similarities, while nodes between clusters are distant from each other.

Here, we only focus on clustering the hub nodes. This is because, attribute nodes are only attribute values of hub nodes, which are helpful for clustering hub nodes. As an example, in Fig. 1(b), the hub nodes (i.e., papers) can be clustered into two sets, i.e., $\{P_1, P_2\}$ and $\{P_3, P_4, P_5\}$.

3.3 Personalized PageRank

To consider both structural and attribute similarities, we utilize Personalized PageRank (PPR) [45], [46] to measure the similarity between two nodes. This is possible because attribute values are modeled as nodes in the star-schema heterogeneous graph. Further, PPR is among the most popular node similarity measures. PPR is proposed based on random walks. A random walk starts at a source node s. At each step in a random walk, PPR either restarts the random walk with probability α or selects an outgoing node with probability $1 - \alpha$. Let $\mathbf{P}$ be a transition probability matrix, where $p[u,v]$ in $\mathbf{P}$ denotes the probability of the outgoing node v from the current node u, i.e., $\mathbf{P}$ is used to choose the outgoing node of a random walk. Specifically, $p[u,v]$ can be computed as:

$$p[v,u] = \begin{cases} \frac{w(v,u)}{1+c} & v \in \mathcal{A}_0 \\ w(v,u) & v \notin \mathcal{A}_0 \end{cases} \tag{3}$$

Here, the weight for all types of nodes (i.e., 1 + c) is used to normalize the probability for hub nodes. In Fig. 1(b), let c_i $(0 \leq i \leq 3)$ be initialized to 1. For hub node P_2, $p(P_2,P_1) = 0.125$, $p(P_2,P_3) = 0.125$, $p(P_2,U_1) = 0.125$, $p(P_2,U_2) = 0.125$, $p(P_2,T_2) = 0.25$, and $p(P_2,C_1) = 0.25$. For attribute node U_1, $p(U_1,P_1) = 0.5$ and $p(U_1,P_2) = 0.5$.

Let a $|V| \times 1$ vector e_s denote a source node s, where $e_s[s] = 1$ and $e_s[v] = 0$ $(v \in V \wedge v \neq s)$. The recursive PPR propagates similarity scores until convergence, i.e., the PPR score from source node s to any node in V is computed as:

$$\begin{aligned} \pi(s,\star) &= \alpha e_s + (1-\alpha)\mathbf{P^T}\pi(s,\star) \\ &= \alpha(\mathbf{I} - (1-\alpha)\mathbf{P^T})^{-1} e_s \\ &= \alpha \sum_{i=0}^{\infty} (1-\alpha)^i (\mathbf{P^T})^i e_s \end{aligned} \tag{4}$$

Many proposals on efficient PPR score computation exist, including matrix based methods [46], [47], [48] and Monte Carlo based approaches [49], [50]. We utilize a backward search method (BSA) [45] to compute the PPR scores from every node v $(\in V)$ to a target node t that satisfies the absolute approximation guarantee. We choose BSA because (i) it is a simple yet fast method; and (ii) DBSCAN only aims at large PPR scores above a threshold, and thus, an absolute error bound (e.g., 10^{-3}) is appropriate when using DBSCAN.

BSA starts from the target node t, and propagates information along the reverse direction of the edges. The search iteratively updates two properties for each node v, i.e., residue $r(v,t)$ and reserve $\hat{\pi}(v,t)$. The former represents the information to be propagated to other nodes, and the latter denotes the estimated PPR value of target t with respect to node v.

Algorithm 1 presents the pseudo-code of BSA. It takes as inputs a graph G, a target node t, a restart probability α, and an error bound r_{max}, and outputs $\hat{\pi}(v,t)$ for all $v \in V$. First of all, BSA sets residue $r(t,t) = 1$ and residue $r(v,t) = 0$ where $v \in V$ and $v \neq t$ (line 1). Then, it initializes every node v in G to have reserve $\hat{\pi}(v,t) = 0$ (line 2). Next, a while loop is performed until every node $v \in V$ satisfying $r(v,t) \leq r_{max}$ (lines 3–8). In each iteration, BSA picks any node $v \in V$ with $r(v,t) > r_{max}$ (line 4), and it updates all $r(u,t)$ for nodes u belonged to in-edges (u,v) (i.e., $h(u,v) = 1$) (lines 5–6). Then, the algorithm converts

Algorithm 1: Backward Search Algorithm

Input: a graph $G = (V, E, \varphi, \psi, w)$, a target node t, a restart probability α, an error bound r_{max}
Output: $\hat{\pi}(v,t)$ for all $v \in V$

1 $r(t,t) \leftarrow 1, r(v,t) \leftarrow 0$ for all $v \neq t$
2 $\hat{\pi}(v,t) \leftarrow 0$ for all $v \in V$
3 **while** $\exists v \in V$ *having* $r(v,t) > r_{max}$ **do**
4 pick any node $v \in V$ with $r(v,t) > r_{max}$
5 **for** *each node* $u \in V$ *having* $h(u,v) = 1$ **do**
6 $r(u,t) \leftarrow r(u,t) + (1-\alpha) \cdot r(v,t) \cdot p[u,v]$
7 $\hat{\pi}(v,t) \leftarrow \hat{\pi}(v,t) + \alpha \cdot r(v,t)$
8 $r(v,t) \leftarrow 0$
9 **return** $\hat{\pi}(v,t)$ *for all* $v \in V$

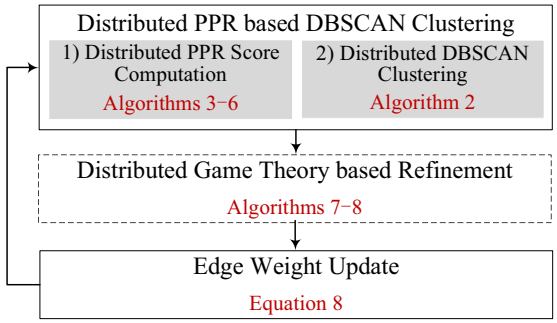


Fig. 2. Framework of Clustering Algorithms

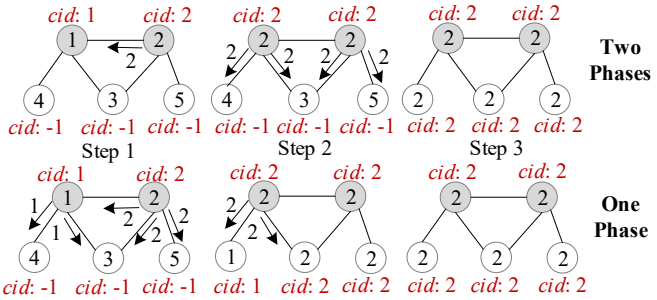


Fig. 3. The Example of Distributed DBSCAN

the α portion of residue $r(v,t)$ to the reserve $\hat{\pi}(v,t)$ (line 7), and it updates the residue $r(v,t)$ to 0 (line 8). Finally, BSA returns all $\hat{\pi}(v,t)$ $(v \in V)$ (line 9).

PPR score is computation asymmetric. For any two nodes u and v in a heterogeneous graph, we can get that $\pi(u,v) \neq \pi(v,u)$. As stated in [51], for any two nodes in the same cluster, the symmetric similarity is required. Motivated by this, we define the **distance between two nodes** in heterogeneous graph $d(u,v) = max\{\pi(u,v), \pi(v,u)\}$. Note that, the higher PPR score is, the more similar between two nodes.

4 Framework of Clustering

In this section, we first describe the framework of clustering algorithms, and then present our edge weight update strategy.

4.1 Clustering Framework

In order to improve the scalability of clustering, a distributed underlying processing system is needed. We choose Blogel, a popular and efficient graph processing distributed system, where vertex-centric distributed computation mode is used for graph clustering. However, GVD partitioner in Blogel can divide the whole graph into different workers, where adjacent nodes are distributed to be at the same worker. Therefore, the communication cost (i.e., the number of messages) between workers drops a lot.

Fig. 2 illustrates the overview of our framework. Similar as the previous work [13], our algorithms are iterative so that the attribute weights can be updated iteratively based on the clustering result. Each iteration contains three phases, i.e., distributed PPR based DBSCAN clustering, distributed game theory based refinement, and edge weight update, where the first phase contains two steps, i.e., distributed PPR score computation and distributed DBSCAN clustering. Hence, we have four steps in total every iteration, that is, we i) first compute PPR scores to measure the similarity of nodes, ii) cluster the nodes according to PPR scores, iii) refine the clustering result, and iv) update the edge weights accordingly. Note that, the third step (i.e., refinement) is optional that further improves the result quality using game theory.

Distributed PPR Score Computation: PPR score computation is costly. Based on Blogel, we present a distributed version of BSA (discussed in Section 3.3), and three distributed incremental approaches (i.e., Partial PPR Algorithm, Approximate PPR Algorithm, and Sampling PPR Algorithm) that aim to enable efficient PPR score computation when edge weights are updated. The corresponding methods (Algorithms 3–6) will be detailed in Section 5. The obtained PPR score is utilized to measure the similarity between two nodes.

Distributed DBSCAN Clustering: After obtaining the PPR scores between nodes in the heterogeneous graph, we propose an improved distributed DBSCAN Algorithm (Algorithm 2) to detect the clusters, as to be described in the following subsection 4.2.

Distributed Game Theory based Refinement: The quality of clustering can be further improved. Clustering can also be regarded as a strategic game [52], [53]. Thus, the result of DBSCAN method can be refined by the game theory. We present a distributed game theory based method, as the corresponding method (Algorithms 7–8) is to be presented in Section 6.

Edge Weight Update: As stated in Section 3.2, edge weights can be computed by using Equation 2, which depends on weight constants c_i $(0 \leq i \leq |\mathcal{A}| - 1)$. Some attribute types contribute more to the quality of clustering, and thus, the weight constants can be updated according to the clusters obtained in every iteration. The detailed update method is discussed in the following subsection 4.3.

4.2 Distributed DBSCAN Clustering

DBSCAN is a popular density-based clustering method. It relies on two parameters to characterize density or sparsity, i.e., a positive real value δ and a positive integer $minPts$. Next, we introduce the definitions of core node and density reachable node to capture the density.

Definition 5. **(Core node)** A node u is a core node if at least $minPts$ nodes v (including u itself) satisfy $d(u,v) \geq \delta$, where $d(u,v)$ denotes the similarity between u and v.

Definition 6. **(Density reachable node)** A node u is a density reachable from node v if there exist a sequence of nodes $x_1, x_2, ..., x_t (t \geq 2)$ such that (i) $x_1 = v$ and $x_t = u$; (ii) x_i $(1 \leq i \leq t-1)$ are core nodes; and (iii) $d(x_i, x_{i+1}) \geq \delta$ $(1 \leq i \leq t-1)$.

Based on Definitions 5 and 6, we design a distributed DBSCAN algorithm to obtain clusters. Inspired by existing distributed DBSCAN method NG-DBSCAN [17], we can construct a new graph called δ-graph on the hub nodes in the heterogeneous graph to facilitate vertex-centric distributed DBSCAN clustering. More specifically, if the distance between the hub node u and the hub node v, $d(u,v) \geq \delta$, there exists an edge in δ-graph to connect u and v. Hence, δ-graph connects nodes with high similarity.

Based on Blogel, every node in the δ-graph can be processed parallelly, and every node can send messages to its connected/neighbor nodes. Algorithm 2 depicts the pseudo-code of Distributed DBSCAN Algorithm. It takes δ-graph and $minPts$ as inputs. Initially, the algorithm sets *cid*s (the cluster id) of core nodes to their unique non-negative *vertexId*, and sets *cid*s of non-core nodes to -1 (lines 1–5). Then, a while loop is performed until

Algorithm 2: Distributed DBSCAN Algorithm

Input: a δ-graph $G' = (V, E)$, a positive integer $minPts$
Output: $cid(v)$ for all nodes $v \in G'$

1 **for** *each node* v *in* G **parallelly do**
2 **if** $|\{\forall u \in V(h(u,v) = 1)\}| \geq minPts - 1$ **then**
3 $v.cid = v.vertexId$
4 **else**
5 $v.cid = -1$
6 **while** *active node exists* **do**
7 **for** *each active node* v **parallelly do**
8 $msg_v \leftarrow message(v)$
9 $v.cid = max(v.cid, msg_v)$
10 **for** *each core node* v **parallelly do**
11 **for** *each core node* u **parallelly do**
12 **if** $h(u,v) = 1 \wedge v.vertexId > u.vertexId$ **then**
13 $msg = v.cid$
14 send msg to node u
15 **while** *active node exists* **do**
16 **for** *each active node* v **parallelly do**
17 $msg_v \leftarrow message(v)$
18 $v.cid = max(v.cid, msg_v)$
19 **for** *each core node* v **parallelly do**
20 **for** *each non-core node* u **parallelly do**
21 **if** $h(u,v) = 1$ **then**
22 $msg = v.cid$
23 send msg to node u
24 **return** $cid(v)$ *for all nodes* $v \in G'$

no node receives message (i.e., no active node exists) (lines 6–14). In each iteration, each active node updates its *cid* to the largest number among the received messages and its original *cid* (lines 7–9). After that, each core node sends its *cid* to the neighbor nodes if its *vertexID* is bigger than that of neighbor node (lines 10–14). Next, another while loop is performed until no active node exists (lines 15–23). The difference between these two loops is that, core nodes send messages to other core nodes in the former loop, while messages are sent between core nodes and non-core nodes in the latter loop. Finally, DBSCAN returns *cid*s of all nodes (line 24).

Different from NG-DBSCAN that detects clusters among all the nodes in one-phase, our method is divided into two phases (i.e., two while loops) which first handles core nodes and then handles density reachable nodes. This optimization reduces the number of the messages sent between nodes, and thus can improve DBSCAN efficiency. A example is shown in Fig. 3, where the grey nodes are core nodes and white nodes are density reachable nodes. The digits in the nodes denote the *vertexId*s, the texts in red are *cid*s (i.e., cluster ids) of nodes, and the black arrows with digits represent the messages. In this example, two-phases processing needs 5 messages but one-phase processing needs 7 messages in total. Hence, our method is more efficient.

4.3 Edge Weight Update

As discussed in Section 3.2, we fix c_0 at 1 for the hub type of nodes (e.g., papers in a bibliographic network). However, the weight constant c_i for the specific attribute type $\mathcal{A}_i$ is initialized to 1 and then is updated iteratively. Similar as [14], c_i in the $(y+1)^{th}$ iteration is computed as:

$$c_i^{y+1} = \frac{1}{2}(c_i^y + \Delta c_i^y) \tag{5}$$

To accurately determine the extent of weight increment Δc_i^y, a majority mechanism can be used: if a large portion of nodes within clusters share the same value of a certain attribute type $\mathcal{A}_i$, $\mathcal{A}_i$ has a good clustering tendency, and thus, the weight constant c_i of $\mathcal{A}_i$ is increased. On the other hand, if nodes within clusters have a random distribution on values of a certain attribute type $\mathcal{A}_i$, $\mathcal{A}_i$ is not a good clustering attribute, and hence, the corresponding weight constant c_i should be decreased. Thus, the entropy of each attribute type in y^{th} iteration can be used to define the weight increment Δc_i^y.

Assume that the node set V in a heterogeneous graph G can be clustered into k parts V_j ($1 \leq j \leq k \wedge |V_j| \geq 2$) in the y^{th} iteration. As DBSCAN can find outliers, we only focus on clusters V_j with $|V_j| \geq 2$. Based on this, the entropy of a certain attribute type $\mathcal{A}_i$ on k clusters is defined as:

$$\textit{entropy}(\mathcal{A}_i) = \sum_{j=1}^{k} \frac{|V_j|}{\sum_{x=1}^{k} |V_x|} \sum_{x=1}^{|\mathcal{A}_i|} (-p_{xj} log_2 p_{xj}) \tag{6}$$

where p_{xj} denotes the percentage of nodes in cluster V_j having the same attribute value $a_x \in \mathcal{A}_i$, and $|\mathcal{A}_i|$ represents the total number of attribute values in a certain attribute type $\mathcal{A}_i$. According to the definition of entropy, the smaller $\textit{entropy}(\mathcal{A}_i)$ is, the better the clustering attribute $\mathcal{A}_i$ is. Hence, the increment Δc_i^y is defined as:

$$\Delta c_i^y = \frac{\frac{c}{\textit{entropy}(\mathcal{A}_i)}}{\sum_{j=1}^{|\mathcal{A}|-1} \frac{1}{\textit{entropy}(\mathcal{A}_j)}} \tag{7}$$

Equation 7 can ensure $\sum_{i=1}^{|\mathcal{A}|-1} c_i^{y+1} = c$ after weight constant updates. To sum up, we can get that

$$c_i^{y+1} = \frac{1}{2}(c_i^y + \frac{c}{\sum_{j=1}^{|\mathcal{A}|-1} \frac{\textit{entropy}(\mathcal{A}_i)}{\textit{entropy}(\mathcal{A}_j)}}) \tag{8}$$

After each c_i^{y+1} updates, PPR scores will be updated accordingly. To compute PPR scores efficiently, we develop incremental methods to avoid unnecessary computation at the next section.

4.4 Discussions

Overall, there are five common parameters used in our proposed framework: i) c used to control the total weight of all the attribute types; ii) δ and $minPts$ used for DBSCAN; and iii) restart probability α and an error bound r_{max} for PPR computation. The optimal values of c, δ, and $minPts$ depend on the distribution of dataset (as verified via experiments in Section 7), α is set to 0.2 by following previous studies [49], [50], and r_{max} is set to 10^{-3} to ensure the accuracy of PPR scores.

5 Incremental PPR Computation

We detail distributed BSA algorithm and three distributed incremental PPR computation approaches that aim to reduce the cost of PPR score recomputation after edge weight updates.

Algorithm 3: Distributed Backward Search Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, a restart probability α, an error bound r_{max}
Output: $\hat{\Pi}_v$ for all $v \in V$

1 **for** *each node* $v \in V$ **parallelly do**
2 **if** v *is a hub node* **then**
3 insert $(v, 1)$ in R_v
4 **while** *active node exists* **do**
5 **for** *each active node* v **parallelly do**
6 $msg_v \leftarrow message(v)$
7 $R_v \leftarrow R_v + : + msg_v$
8 **for** *each node* v *st.* $(\exists t, R_v[t] > r_{max})$ **parallelly do**
9 **for** *each node* u *having* $h(u, v) = 1$ **parallelly do**
10 $msg \leftarrow (1-\alpha) \cdot p[u, v] \cdot R_v[t] (\forall t, R_v[t] > r_{max})$
11 send msg to node u
12 $R_v \leftarrow R_v[t]\ (\forall t, R_v[t] \leq r_{max})$
13 $\hat{\Pi}_v \leftarrow \hat{\Pi}_v + : + \alpha \cdot R_v[t] (\forall t, R_v[t] > r_{max})$
14 **return** $\hat{\Pi}_v$ *for all hub nodes* $v \in V$

Algorithm 4: Distributed Partial PPR Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, current iteration y, a restart probability α, an error bound r_{max}
Output: $\hat{\Pi}_v$ for all $v \in V$

1 read R_v^0 and $\hat{\Pi}_v^0$ computed by DPPA for all nodes $v \in V$
2 **for** *each node* $v \in V$ **parallelly do**
3 $R_v \leftarrow R_v^0$
4 $\hat{\Pi}_v \leftarrow \hat{\Pi}_v^0$
5 **while** *active node exists* **do**
6 **for** *each active node* v **parallelly do**
7 $msg_v \leftarrow message(v)$
8 $R_v \leftarrow R_v + : + msg_v$
9 **for** *each node* v *st.* $(\exists t, R_v[t] > r_{max})$ **parallelly do**
10 **for** *each node* u *having* $h(u, v) = 1$ **parallelly do**
11 $msg \leftarrow (1-\alpha) \cdot R_v[t] \cdot p^y[u, v] (\forall t, R_v[t] > r_{max})$
12 send msg to node u
13 $R_v \leftarrow R_v[t]\ (\forall t, R_v[t] \leq r_{max})$
14 $\hat{\Pi}_v \leftarrow \hat{\Pi}_v + : + \alpha \cdot R_v[t] (\forall t, R_v[t] > r_{max})$
15 **return** $\hat{\Pi}_v$ *for all* $v \in V$

5.1 Distributed PPR Computation

Distributed Backward Search Algorithm (DBSA) is a distributed version of BSA (discussed in Section 3.3). The centralized version BSA computes the PPR scores between all nodes to a particular target node t. In contrast, DBSA computes all the PPR scores of all pairs of nodes simultaneously. In order to maintain the reserve and residue values of all pairs of nodes, we create two maps $\hat{\Pi}_v$ and R_v for each node v to store all the reserve and residue values between any node $u \in V$ and v, respectively. Here, we use map because it can avoid storing zero values to reduce the storage complexity, and the maximum size of R_v or $\hat{\Pi}_v$ is $|V|$. DBSA computes PPR scores iteratively. In each iteration, each node v propagates information to its neighbor nodes, and updates its two maps, i.e., the residue map R_v and the reserve map $\hat{\Pi}_v$. Here, instead of propagating one qualified $r(v, u)$ value each time by BSA, DBSA will propagate all qualified $r(v, u)$ values at the same time, where a qualified value means $r(v, u) > r_{max}$.

Algorithm 3 presents the pseudo-code of DBSA. It takes as inputs a heterogeneous graph G, a restart probability α, and an error bound r_{max}, and outputs $\hat{\Pi}_v$ for all nodes $v \in V$. First of all, DBSA parallelly inserts $(v, 1)$ in the residue map R_v if v is a hub node (lines 1–3). Here, only $R_v[v]$ of the hub nodes v is set to 1 as we only cluster hub nodes. Next, a while loop is performed until no active node exists (lines 4–13). In each iteration, each active node v updates its residue map R_v parallelly according to messages received from its neighbor nodes (lines 5–7). Here, "+:+" denotes the add operation on two maps, as the message is also denoted as a map. More specifically, for any key exists in two maps, we sum up two corresponding values; while for any key exists in only one map, we will first create a new entry (key, 0) in the other map and then sum up two values. Then, each node v having any t satisfying $R_v[t] > r_{max}$ sends messages to its neighbor nodes (lines 9–11), and updates its two maps corresponding (lines 12–13). Finally, it returns $\hat{\Pi}_v$ for all nodes $v \in V$ (line 14).

5.2 Distributed Partial PPR Computation

According to DBSA, the PPR score depends on $p[u, v]$, which changes when a weight update occurs. According to Equation 3, the transition probability $p^y[u, v]$ in the y^{th} iteration is defined as:

$$p^y[u, v] = \begin{cases} \frac{c_i^y \cdot h(u,v)}{c_i^0 \cdot (1+c) \cdot \sum_{x \in \mathcal{A}_i} h(u,x)} & u \in \mathcal{A}_0 \wedge v \in \mathcal{A}_i \\ \frac{h(u,v)}{\sum_{x \in V} h(u,x)} & u \notin \mathcal{A}_0 \end{cases} \quad (9)$$

According to Equation 9, for a hub node u and any node v (belonging to $\mathcal{A}_i$), $p^y[u, v] = \frac{c_i^y}{c_i^0} p^0[u, v]$; and for an attribute node u and any node v, $p^y[u, v] = p^0[u, v]$. Motivated by this, we present a Distributed Partial PPR Algorithm (DPPA), which only processes the hub nodes instead of the whole graph nodes in advance, and stores the corresponding residues and reserves of all nodes for later reuse.

DPBSA is similar to DBSA, and thus, its pseudo-code is omitted. The only difference between DPBSA and DBSA is that, in line 9 of Algorithm 3, DPBSA only processes hub nodes while DBSA processes all graph nodes, i.e., DPBSA only propagates the residue R_v of hub nodes v to other nodes. Finally, we store non-empty R_v and $\hat{\Pi}_v$ for all nodes $v \in V$ that can be used in PPR score recomputation.

Based on DPBSA, a Distributed Partial PPR Algorithm (DPPA) is developed, with the corresponding pseudo-code shown in Algorithm 4. First, DPPA reads $\hat{\Pi}_v^0$ and S_v^0 computed by DPBSA, and then initializes $\hat{\Pi}_v$ and S_v for all $v \in V$ (lines 1–4). Thereafter, the processing is the same as lines 5–15 of Algorithm 3. Hence, by applying DPBSA once, the pre-computed scores can be reused by DPPA in every iteration after edge weight updates.

5.3 Distributed Approximate PPR Computation

Following the basic idea of an existing approach [54], we keep the reserve for every node (computed by BSA) unchanged but update the residue (computed by BSA) for each node, to approximately simulate edge weight updates. In the following, we derive how to update the residue for every node.

According to the derivations in our previous work [19], in the y^{th} and $(y+1)^{th}$ iteration, for each hub node s,

$$\hat{\pi}^y(s, t) + \alpha r^y(s, t) = \alpha e_t^{-1}[s] + (1-\alpha) \sum_{i=0}^{|\mathcal{A}|-1} \sum_{x \in \mathcal{A}_i} \frac{c_i^y \cdot h(s, x)}{(1+c) \cdot \sum_{u \in \mathcal{A}_i} h(s, u)} \cdot \hat{\pi}^y(x, t) \quad (10)$$

Algorithm 5: Distributed Approximate PPR Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, current iteration y, a restart probability α, an error bound r_{max}
Output: $\hat{\Pi}_v$ for all $v \in V$

1 read R_v^0 and Π_v^0 computed by DPBSA for all $v \in V$
2 **for** *each node* $v \in V$ **parallelly do**
3 $\quad \hat{\Pi}_v \leftarrow \Pi_v^0$
4 $\quad$ **if** v *is hub node* **then**
5 $\quad\quad$ **for** *each* $u \in V$ **parallelly do**
6 $\quad\quad\quad R_v^y[u] \leftarrow R_v^0[u] + \sum_{i=0}^{|\mathcal{A}|-1} \frac{(1-\alpha)(c_i^y - c_i^0)}{\alpha(1+c)} \frac{\sum_{x \in \mathcal{A}_i} h(v,x) \cdot \hat{\pi}^0(x,u)}{\sum_{t \in \mathcal{A}_i} h(v,t)}$
7 $\quad$ **else**
8 $\quad\quad R_v^y \leftarrow R_v^0$
9 **while** *active node exists* **do**
10 $\quad$ **for** *each active node* v **parallelly do**
11 $\quad\quad msg_v \leftarrow message(v)$
12 $\quad\quad R_v \leftarrow R_v + : + msg_v$
13 $\quad$ **for** *each node* v *st.* $(\exists t, R_v[t] > r_{max})$ **parallelly do**
14 $\quad\quad$ **for** *each node* u *having* $h(u,v) = 1$ **parallelly do**
15 $\quad\quad\quad msg \leftarrow (1-\alpha) \cdot R_v[t] \cdot p^y[u,v] (\forall t, R_v[t] > r_{max})$
16 $\quad\quad\quad$ send msg to node u
17 $\quad\quad R_v \leftarrow R_v[t]\ (\forall t, R_v[t] \leq r_{max})$
18 $\quad\quad \hat{\Pi}_v \leftarrow \hat{\Pi}_v + : + \alpha \cdot R_v[t] (\forall t, R_v[t] > r_{max})$
19 **return** $\hat{\Pi}_v$ *for all* $v \in V$

$$\hat{\pi}^{y+1}(s,t) + \alpha r^{y+1}(s,t) = \alpha e_t^{-1}[s] + (1-\alpha) \sum_{i=0}^{|\mathcal{A}|-1} \sum_{x \in \mathcal{A}_i} \frac{c_i^{y+1} \cdot h(s,x)}{(1+c) \cdot \sum_{u \in \mathcal{A}_i} h(s,u)} \cdot \hat{\pi}^{y+1}(x,t) \quad (11)$$

By combining Equations 10 and 11 as well as assuming that $\hat{\pi}^y(s,t) = \hat{\pi}^{y+1}(s,t)$, we can get that:

$$r^{y+1}(s,t) = r^y(s,t) + \sum_{i=0}^{|\mathcal{A}|-1} \frac{(1-\alpha)(c_i^{y+1} - c_i^y)}{\alpha(1+c)} \frac{\sum_{x \in \mathcal{A}_i} h(s,x) \cdot \hat{\pi}^y(x,t)}{\sum_{u \in \mathcal{A}_i} h(s,u)} \quad (12)$$

Based on Equation 12, we propose an Distributed Approximate PPR Algorithm (DAPA). In the first iteration, DPBSA (discussed in Section 5.2) is run to store R_v and $\hat{\Pi}_v$ for all $v \in V$ for reuse. Algorithm 5 shows the pseudo-code of DAPA for iterations $y \geq 1$. It first reads R_v^0 and $\hat{\Pi}_v^0$ for all $v \in V$. Then, $\hat{\Pi}_v$ is initialized as $\hat{\Pi}_v^0$ for every $v \in V$. According to Equation 12, $R_v^y[u]$ is parallelly initialized to $r_v^y[u] + \sum_{i=0}^{|\mathcal{A}|-1} \frac{(1-\alpha)(c_i^y - c_i^0)}{\alpha(1+c)} \frac{\sum_{x \in \mathcal{A}_i} h(s,x) \cdot \hat{\pi}^0(x,t)}{\sum_{u \in \mathcal{A}_i} h(s,u)}$ for each hub node $v \in \mathcal{A}_0$ (lines 4–6), while R_v^y is initialized to R_v^y for each attribute node v (lines 7–8). In the sequel, the processing (lines 9–19) is the same as lines 5–15 of Algorithm 3.

5.4 Distributed Sampling PPR Computation

The previous two methods DPPA and DAPA aim to reduce the cost of PPR score recomputation after updating the edge weights by partially storing the pre-computed scores for reuse and by approximately updating PPR scores to simulate the edge weight updates. Furthermore, we can reduce the communication cost among all the nodes to improve the efficiency of PPR score recomputation. According to DBSA, the information/message sent from node v to node u is below:

$$msg(u \leftarrow v) = (1-\alpha) \cdot R_v[u] \cdot p[u,v] \quad (13)$$

Algorithm 6: Distributed Sampling PPR Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, current iteration y, a restart probability α, an error bound r_{max}, a sampling bound p_{max}, a sampling rate r_s
Output: $\hat{\Pi}_v$ for all $v \in V$

1 read R_v^0 and $\hat{\Pi}_v^0$ computed by DPBSA for all $v \in V$
2 **for** *each node* $v \in V$ **parallelly do**
3 $\quad R_v \leftarrow R_v^0$
4 $\quad \hat{\Pi}_v \leftarrow \hat{\Pi}_v^0$
5 **while** *active node exists* **do**
6 $\quad$ **for** *each active node* v **parallelly do**
7 $\quad\quad msg_v \leftarrow message(v)$
8 $\quad\quad R_v \leftarrow R_v + : + msg_v$
9 $\quad$ **for** *each node* v *st.* $(\exists t, R_v[t] > r_{max})$ **parallelly do**
10 $\quad\quad$ **for** *each node* u *having* $h(u,v) = 1$ **parallelly do**
11 $\quad\quad\quad$ **if** $p^y[u,v] \leq p_{max} \wedge rand()\%100 > 100 r_s$ **then**
12 $\quad\quad\quad\quad$ continue
13 $\quad\quad\quad msg \leftarrow (1-\alpha) \cdot R_v[t] \cdot p^y[u,v] (\forall t, R_v[t] > r_{max})$
14 $\quad\quad\quad$ send msg to node u
15 $\quad\quad R_v \leftarrow R_v[t]\ (\forall t, R_v[t] \leq r_{max})$
16 $\quad\quad \hat{\Pi}_v \leftarrow \hat{\Pi}_v + : + \alpha \cdot R_v[t] (\forall t, R_v[t] > r_{max})$
17 **return** $\hat{\Pi}_v$ *for all* $v \in V$

According to Equation 13, the larger $p[u,v]$ is, the more information the node u receives. Moreover, the information sent from node v drops exponentially as τ (i.e. the number of steps of random walk) increases. As a result, when $p[u,v]$ is small, the information transmission between node u and node v has little influence on the final PPR scores. Consider the situation that a node v is connected with massive nodes. For example, in DBLP dataset, a conference attribute node is connected to massive paper nodes. In this case, $p[\star, v]$ is extremely small, and thus, the information transmission has little impact on the final result. Motivated by this, we present a Distributed Sampling PPR Algorithm (DSPA), which samples the messages between node u and node v, if their transition probability is small. Specifically, given a sampling bound p_{max} and a sample rate r_s, if $p[u,v] < p_{max}$, the message will be sent from node v to node u with probability r_s. Note that, $r_s \in (0, 1]$ is used to trade quality for efficiency, as verified in Section 7. In this paper, we set p_{max} to 10^{-3} as default. This is because, if $p[u,v] \leq 10^{-3}$, then $msg(u \leftarrow v) < 10^{-3} = r_{max}$. In other words, if p_{max} is set to 10^{-3}, then the corresponding information transmission has little impact on the result PPR score.

Algorithm 6 depicts the pseudo-code of DSPA. The only difference between Algorithm 4 and Algorithm 6 is that, Algorithm 6 samples before sending messages between node u and node v (lines 11–12), in order to further reduce the communication cost.

6 Game Theory Based Approach

As discussed in Section 4, the game theory technique can be used to further improve the quality of the clustering obtained by DBSCAN. Here, we introduce an objective function, and then, we present a detailed algorithm.

6.1 Objective Function

The basic idea of game theory is that, in strategic games, players compete for same resources to optimize their individual objective functions. Specifically, each player chooses a strategy to minimize

Algorithm 7: Distributed Initial Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, a parameter β in range $[0, 1]$, $\Pi_v(v \in V)$ that stores non-zero distances between v and other nodes

1 create a map num to store the size of every cluster
2 **for** *each node* $v \in V$ **parallelly do**
3 **if** $v.cid \notin num$ **then**
4 insert $(v.cid, 0)$ in num
5 $num[v.cid] \leftarrow num[v.cid] + 1$
6 **for** *each node* $v \in V$ **parallelly do**
7 **for** *each other node* $u \in \Pi_v \wedge u.cid = v.cid$ **do**
8 $ACost_v[u.cid] += \Pi_v[u]$
9 **for** *each node* u *having* $h(u, v) = 1 \wedge u.cid \neq v.cid$ **do**
10 $SCost_v[u.cid] += \frac{1}{2} \cdot w[v, w]$
11 $Cost_v \leftarrow (1 - \beta) \cdot SCost_v - : -\beta \cdot \frac{ACost_v}{num[v.cid]-1}$
12 $cid_{min} \leftarrow \arg\min_x Cost_v[x]$
13 **if** $v.cid \neq cid_{min}$ **then**
14 $cid_{pre} \leftarrow v.cid$, $v.cid \leftarrow cid_{min}$
15 **for** *each* u *st.* $u \in \Pi_v \vee h(u, v) = 1$ **parallelly do**
16 $msg \leftarrow (v, v.cid, cid_{pre})$
17 send msg to node u

his/her own cost without taking into account the effect of his/her choice on other players' objectives. A strategic game has a pure Nash equilibrium, i.e., no player has an incentive to deviate from his/her current strategy.

In our setting, each node can be regarded as a player. In order to apply the above idea, it is important to design an effective objective function. According to Definition 2, each node should be (i) similar to the other nodes in the same cluster, but (ii) dissimilar to nodes in different clusters. Hence, the first part of the objective function for each node v in a cluster V_i can be defined based on PPR scores.

$$\text{AssignmentCost}(v, V_i) = \frac{1}{|V_i| - 1} \sum_{u \in V_i - \{v\}} d(v, u) \quad (14)$$

Obviously, the larger $\text{AssignmentCost}(v, V_i)$ is, the more similar v is to other nodes in cluster V_i. To take into account the dissimilarity to nodes in different clusters, we utilize structure property instead of PPR scores. We do this because the sum of PPR scores from node v to other nodes $u \in V$ equals 1. Based on this, the second part of the objective function for each node v in a cluster V_i can be defined below.

$$\text{StructureCost}(v, V_i) = \sum_{u \notin V_i} \frac{1}{2} w(v, u) \quad (15)$$

Clearly, the larger $\text{StructureCost}(v, V_i)$, the more dissimilar between node v and nodes in other clusters $V_j \neq V_i$.

We combine the two costs, and use a parameter β in range $[0, 1]$ to control their relative importance. Thus, the objective function of a node is defined as:

$$\begin{aligned} \text{cost}(v, V_i) =& (1 - \beta) \cdot \text{StructureCost}(v, V_i) \\ & - \beta \cdot \text{AssignmentCost}(v, V_i) \end{aligned} \quad (16)$$

where β is usually set to 0.5 to balance AssignmentCost and StructureCost.

Algorithm 8: Distributed Game Theory Algorithm

Input: a heterogeneous graph $G(V, E, \varphi, \psi, w)$, a parameter β in range $[0, 1]$, $\Pi_v(v \in V)$ that stores non-zero distances between v and other nodes
Output: $v.cid$ for every node $v \in V$

1 **while** *active node exists* **do**
2 **for** *each active node* $v \in V$ **parallelly do**
3 $msg_v = \{t, t.cid, t.cid_{pre}\} \leftarrow message(v)$
4 $num[t.cid] = num[t.cid] + 1$
5 $num[t.cid_{pre}] = num[t.cid_{pre}] - 1$
6 **if** $t \in \Pi_v$ **then**
7 $ACost_v[t.cid_{pre}] -= S_v[t]$
8 $ACost_v[t.cid] += S_v[t]$
9 **if** $h(v, t) = 1$ **then**
10 $SCost_v[t.cid_{pre}] -= \frac{1}{2} w[t, v]$
11 $Scost_v[t.cid] += \frac{1}{2} w[t, v]$
12 $Cost_v \leftarrow (1 - \beta) \cdot SCost_v - : -\beta \cdot \frac{ACost_v}{num[v.cid]-1}$
13 $cid_{min} \leftarrow \arg\min_x Cost_v[x]$
14 **if** $v.cid \neq cid_{min}$ **then**
15 $v.cid_{pre} \leftarrow v.cid$, $v.cid \leftarrow cid_{min}$
16 **for** *each* $u \in \Pi_v \vee h(u, v) = 1$ **parallelly do**
17 $msg \leftarrow (v, v.cid, v.cid_{pre})$
18 send msg to node u
19 **return** $v.cid$ *for all* $v \in V$

6.2 Game Theory Algorithm

At first, each node is assigned to a cluster V_i according to the result of DBSCAN, as DBSCAN can find k (> 1) clusters V_i $(1 \leq i \leq k)$. Then, in each iteration, every node v in V_i is re-assigned to a new cluster V_j $(i \neq j)$ with the minimum cost(v, V_j) defined in Equation 16. When a Nash equilibrium exists (i.e., no node will be moved to other cluster), the updated cluster of each node is return.

We use $v.cid$ returned from previous Distributed DBSCAN algorithm to indicate which cluster each node v belongs to. The whole process using game theory to refine the cluster result is divided into two phases. At first, we compute a initial AssignmentCost (ACost), StructureCost (SCost), and Cost for each node according to Equations 14−16, and find the nodes can be refined. Thereafter, we update the corresponding ACost, SCost, and Cost based on the refined nodes until it comes to a Nash equilibrium. Here, the refined node is a node moved from the original cluster to a different cluster with the minimum Cost defined in Equation 16.

Algorithm 7 shows the pseudo-code of the Distributed Initial Algorithm. It takes as inputs a heterogeneous graph G, a parameter β in range [0,1], and $\Pi_v(v \in V)$ that stores non-zero distances between v and other nodes. Initially, it creates a map num to count the size of every cluster (lines 1–5). Then, each node v computes the $Cost_v$ by combining ACost and SCost (lines 6–11), and obtains the corresponding cid_{min} with the minimal cost for each node v (line 12). Next, if $cid_{min} \neq v.cid$ (i.e., the node v can be refined), the algorithm updates $v.cid$, and sends a message to the related nodes (lines 13–17). Here, the related nodes u are neighbors of the refined node v or have non-zero distances $\Pi_v[u]$, to help update the Assignmentcost and StructureCost in future. Hence, each message sent from v contains the related node u, the current node cluster id $u.cid$, and the previous cluster id cid_{pre}.

Algorithm 8 depicts the pseudo-code of Distributed Game Theory Algorithm (DGTA). There are only two differences be-

TABLE 2
Complexity Analysis of Our Framework

Methods		Time Complexity	Space Complexity
Inc-Cluster [14]		O(Ln^3)	O(n^2)
SToC [12]		O($m_h log n_h$)	O($n_h log n_h$)
Our Framework	Distributed PPR Score Computation	O($(n_h + n_t)/n_w$)	O(n^x/n_w) ($1 \leq x \leq 2$)
	Distributed DBSCAN Clustering	O($(n^x + m_c D_p + m_d)/n_w$)	
	Distributed Game Theory based Refinement	O($(n^x + m_h + k n_{active})/n_w$)	
	Edge weight update	O(n_a/n_w)	
	Overall	O($t(n^x + m/r_{max} + m_c D_p + k n_{active})/n_w$)	

tween Algorithm 8 and Algorithm 7. The first one is that Algorithm 8 performs a loop until it comes into a Nash equilibrium (i.e., no message is sent between nodes) (line 1). The second one is that Algorithm 8 updates num, $Acost_v$, $Scost_v$, and $Cost_v$ based on messages received (lines 4–12). Finally, Algorithm 8 returns the cluster result, i.e., $v.cid$ of all $v \in V$ (line 19).

6.3 Discussion

In this subsection, we provide the time complexity and space complexity of our framework compared with existing methods, as shown in Table 2. Note that, Inc-Cluster [14] and SToC [12] are two state-of-the-art attributed graph clustering methods, where L denotes the length of the random walk. Assume that n, n_h, and n_a represent the number of nodes, hub nodes, and attribute nodes, respectively; m, m_h, and m_a are the number of edges to connect all the nodes, to connect hub nodes, and to connect hub nodes and attribute nodes, respectively.

GVD partitioner used in Blogel can achieve good performance of load balance, and thus, we assume that the workload is distributed evenly among the work nodes. Hence, the space complexity of our framework on each work node is $O(n^x/n_w)(1 \leq x \leq 2)$, where n_w denotes the number of workers and n^x represents the number of node pairs with non-zero PPR scores. A lot of zero PPR scores of node pairs exist especially for a large graph, and hence, x is much smaller than 2.

Our framework is iterative and contains four steps. Thus, the overall time complexity of our framework is O($t \times (Cost_1 + Cost_2 + Cost_3 + Cost_4)$), where t denotes the number of iterations and $Cost_i$ ($1 \leq i \leq 4$) for each step is shown below.

$Cost_1$ **for Distributed PPR Score Computation.** The corresponding time complexity on each work node is $O((n_h + n_t)/n_w)$, where n_t denotes the number of information propagations during PPR score computations. We develop four methods for distributed PPR score computations, i.e., DBSA, DPPA, DAPA, and DSPA, and hence, n_t varies for these four methods.

- For DBSA, n_t is O($m/(\alpha r_{max})$) [50], and thus, its time complexity is O($m/(\alpha r_{max} n_w)$).
- For DPPA, it uses DBSA to process hub nodes in advance, and stores the corresponding values for reuse. According to the proof in our previous work [19], DPPA is regarded that part of processing is done in advance. Hence, n_t of DPPA is much smaller than that of DBSA, and the time complexity of DPPA is much smaller than that of DBSA.
- For DAPA, it can provide estimation of the residue and reserve values for every node to simulate edge weight updates according to Equation 12. This holds the potential to reduce the number of information propagations substantially in every iteration, and thus, the time complexity of DAPA is smaller than that of DPPA.
- For DSPA, its n_t falls into the range [O($r_s m/(\alpha r_{max})$), O($m/(\alpha r_{max})$)], where $r_s \in (0, 1]$ denotes the sample rate. Hence, the time complexity of DSPA is much smaller than that of DPPA, especially when r_s is small.

$Cost_2$ **for Distributed DBSCAN clustering.** Assume that D_p equals to the maximum shortest path distance between any two hub nodes, m_c denotes the number of edges that connect core nodes, and m_d represents the number of edges that connect core nodes and density reachable nodes. The corresponding time complexity is O($(n^x + m_c D_p + m_d)/n_w$), where O($n^x/n_w$) is the cost to find the core nodes, O($m_c D_p/n_w$) is the cost to update the cluster *id* of core nodes, and O(m_d/n_w) is the cost to update the cluster *id* of density reachable nodes. However, the real time complexity is much smaller than O($(n^x + m_c D_p + m_d)/n_w$), as the number of iterations is much smaller than D_p when updating the cluster *id* of core nodes.

$Cost_3$ **for Distributed Game Theory based Refinement.** Assume that n_{active} denotes the number of re-assigned nodes before reaching Nash equilibrium, and k is the number of clusters obtained. The time complexity is O($(n^x + m_h + k n_{active})/n_w$), where O($n^x/n_w$) is the cost to compute the initial AssignmentCost, O(m_h/n_w) is the cost to calculate the initial StructureCost, and O($(k n_{active})/n_w$) is the cost to reassign nodes and update AssignmentCost and StructureCost.

$Cost_4$ **for Edge Weight Update.** Since we only update the edges between attribute nodes and hub nodes, its time complexity on each work node is O(m_a/n_w).

As a summary, the overall time complexity of our framework is O($t(n^x + m/r_{max} + m_c D_p + k n_{active})/n_w$). We can see the time and space complexities of our framework are better than those of Inc-Cluster, but worse than SToC. However, the clustering quality of our framework is better than that of SToC, as verified in our previous work [19], where we have compared our centralized methods with Inc-Cluster and SToC. In the presented clustering framework, distributed PPR score computation methods DPPA, DAPA, and DSPA aim to improve the efficiency, while DGTA aims to further enhance the quality. Note that, DGTA is independent of DPPA, DAPA, and DSPA, and thus, DGTA can be applied to any other clustering algorithm in the refinement step.

7 Experimental Evaluation

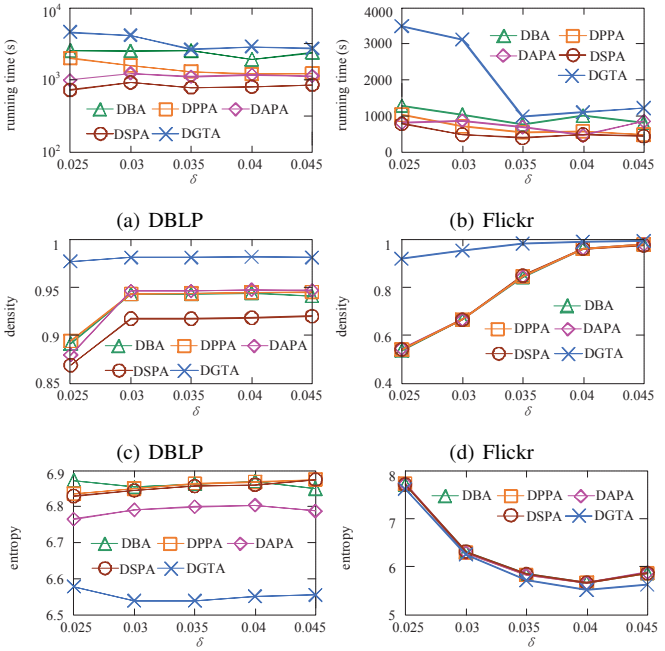
In this section, we proceed to evaluate experimentally the effectiveness, efficiency and scalability of our proposed techniques.

7.1 Experimental Settings

In our experiments, we employ two real-life datasets *DBLP* and *Flickr*, as shown in Table 3. The *DBLP* dataset is extracted from a website http://dblp.uni-trier.de/, which is a typical bibliographic

TABLE 3
Statistics of Two Real-life Datasets

DBLP		
Objects and Relations	**Type**	**Cardinality**
Papers	$\mathcal{A}_0$	236,098
Authors	$\mathcal{A}_1$	734,593
Keywords	$\mathcal{A}_2$	20,704
Publication venues	$\mathcal{A}_3$	405
Edges between papers	$\mathcal{R}_0$	126,855
Edges between papers and authors	$\mathcal{R}_1$	1,257,028
Edges between papers and keywords	$\mathcal{R}_2$	846,283
Edges between papers and venues	$\mathcal{R}_3$	236,098
Flickr		
Objects and Relations	**Type**	**Cardinality**
Images	$\mathcal{A}_0$	260,921
Users	$\mathcal{A}_1$	173,282
Tags	$\mathcal{A}_2$	27,555
Edges between images	$\mathcal{R}_0$	146,523
Edges between images and tags	$\mathcal{R}_1$	272,491
Edges between images and users	$\mathcal{R}_2$	1,278,141



(a) DBLP (b) Flickr (c) DBLP (d) Flickr (e) DBLP (f) Flickr

Fig. 4. Performance of Our Methods vs. δ

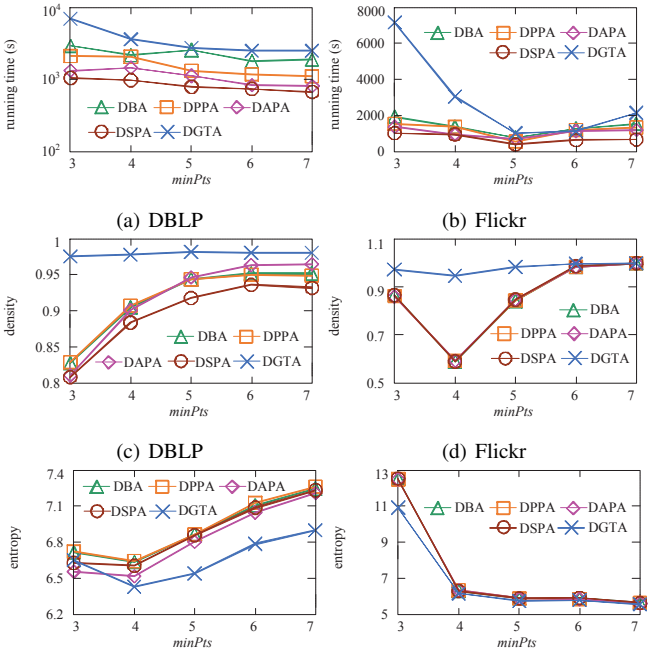
information dataset. The *Flickr* dataset is extracted from a website https://www.flickr.com/, which is a typical social image dataset. *DBLP* contains about 1 million nodes while *Flickr* includes about 0.44 million nodes, where our previous centralized clustering methods [19] cannot run on these two big attributed graphs.

As discussed in Section 2, no previous distributed attributed/heterogeneous graph clustering method is available and can be used for comparisons. Hence, we extend our centralized methods [19] to the distributed ones to improve the scalablity. In our experiments, we adopt a baseline method (denoted as DBA), which uses DBSA (depicted in Algorithm 3) to compute the PPR scores and utilizes Distributed DBSCAN Algorithm (shown in Algorithm 2) to do the clustering.

The efficiency of our clustering methods is reported in terms of the running time. Since large *DBLP* and *Flickr* datasets without ground truths, the effectiveness, i.e., the quality of the generated clusters V_j $(1 \leq j \leq k)$, is measured using density and entropy,

TABLE 4
Parameter Ranges and Default Values

Parameter	**Range**	**Default**
δ $(\times 10^{-3})$	2.5, 3, 3.5, 4, 4.5	3.5
$minPts$	3, 4, 5, 6, 7	5
β	0.2, 0.35, 0.5, 0.65, 0.8	0.5
r_s	0.2, 0.4, 0.6, 0.8, 1	0.6
n_w	4, 6, 8, 10, 12	12
c	0.1875, 0.375, 1.5, 3, 6	3 or 2
p_e $(\times 10^{-5})$	0.125, 0.625, 1.125, 1.625, 2.125	1.125



(a) DBLP (b) Flickr (c) DBLP (d) Flickr (e) DBLP (f) Flickr

Fig. 5. Performance of Our Methods vs. $minPts$

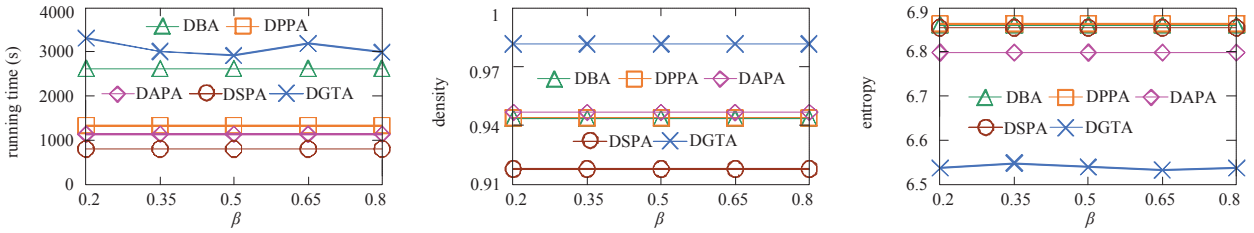
thus following two existing studies [13], [14].

$$density = \frac{\sum_{j=1}^{k} \sum_{v \in V_j, u \in V_j} h(v,u)}{\sum_{v \in \{\cup_{j=1}^{k} V_j\}, u \in \{\cup_{j=1}^{k} V_j\}} h(v,u)} \tag{17}$$

$$entropy = \sum_{i=0}^{|\mathcal{A}|-1} \frac{c_i \sum_{j=1}^{k} (|V_j| \sum_{x=1}^{|\mathcal{A}_i|} (-p_{xj} log_2 p_{xj}))}{c \sum_{j=1}^{k} |V_j|} \tag{18}$$

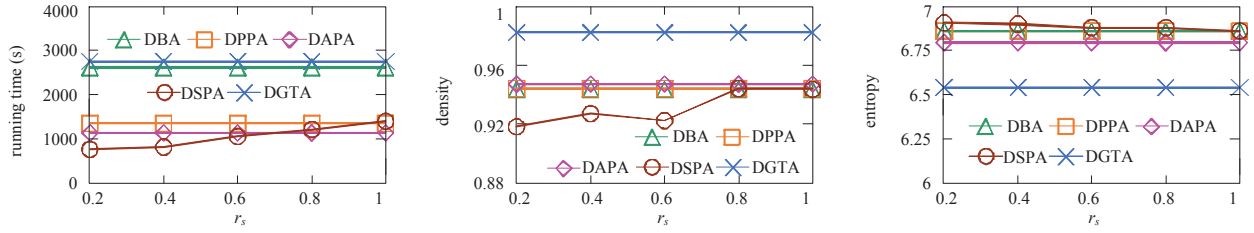
Here, p_{xj} denotes the percentage of nodes in cluster V_j that have the same attribute value $a_x \in \mathcal{A}_i$, and $|\mathcal{A}_i|$ represents the total number of attribute values in $\mathcal{A}_i$. A clustering is good, if its density is high, and its entropy is low.

We evaluate the performance of our clustering approaches under various parameters, as summarized in Table 4. Here, δ and $minPts$ are two DBSCAN parameters, β is used in the game theory objective function, r_s is the sample rate for DSPA, n_w denotes the number of work nodes, c is the weight constant for all attribute types, and p_e is the probability to generated ER random network. Note that, c is set to the number of attribute types (i.e., 2 for *Flickr* and 3 for *DBLP*) as default. In each experiment, we vary one parameter, and fix the others to their default values. All the methods were implemented in C++. We conducted the experiments on a 12-node Dell cluster, where cluster nodes are connected via Gigabit Ethernet. Each node has two 12-core processors (Intel Xeon E5-2640 v4@2.40GHz) and 128GB



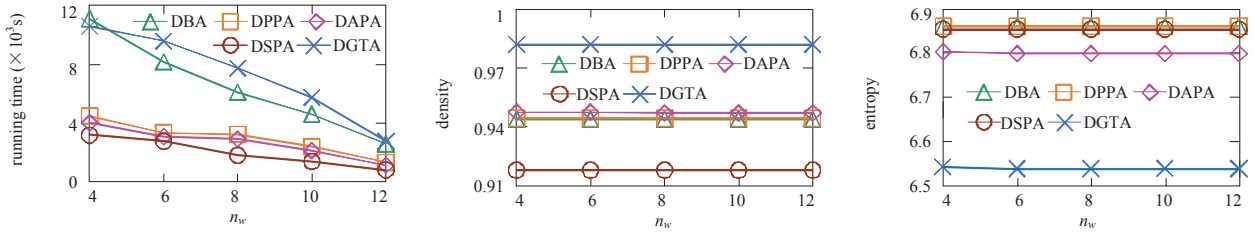
(a) DBLP (b) DBLP (c) DBLP

Fig. 6. Performance of Our Methods vs. β



(a) DBLP (b) DBLP (c) DBLP

Fig. 7. Performance of Our Methods vs. r_s



(a) DBLP (b) DBLP (c) DBLP

Fig. 8. Performance of Our Methods vs. n_w

RAM. Although each work node is a multi-core machine, only one core is used to run our experiments.

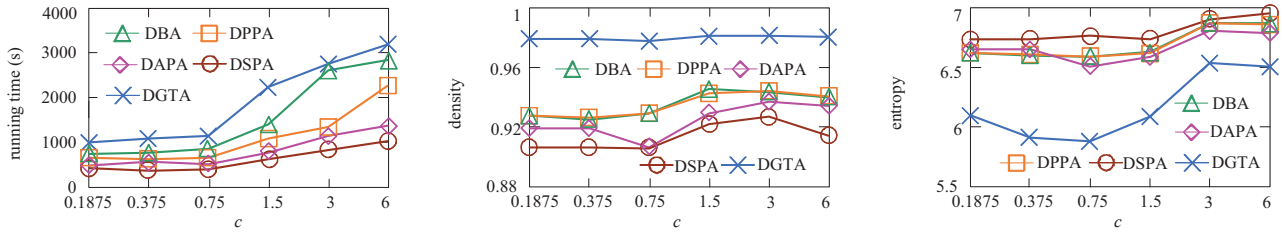
7.2 Performance Study

We investigate the performance of our presented four distributed methods when varying different parameters, including the clustering method that uses the Distributed Partial PPR Algorithm to compute PPR scores (DPPA), the method that uses the Distributed Approximate PPR Algorithm to compute PPR scores (DAPA), the method that utilizes the Distributed Sampling PPR Algorithm to compute PPR scores (DSPA), and the method that utilizes the Distributed Game Theory Algorithm to refine the result (DGTA), compared with the baseline algorithmm (DBA).

First, we study the impact of δ on the efficiency and effectiveness of DBA, DPPA, DAPA, DSPA, and DGTA. Fig. 4 depicts the performance when changing δ on two datasets *DBLP* and *Flickr*. The first observation is that, the efficiency (i.e., the running time) of DGTA is the worst, followed by DBA, DPPA, and DAPA, while DSPA is the best. This is because, DGTA needs additional cost to refine the result, whereas DPPA, DSPA, and DAPA are incremental PPR score computation methods that store residues and reserves of nodes for reuse, which eliminates substantial computation costs. In addition, DSPA samples the messages propagated between nodes, resulting in better performance in terms of efficiency. However, the running time of DGTA decreases rapidly with the growth of δ on *Flickr*, while others are affected slightly. The possible reason is that, as δ increases, fewer nodes are included in clusters, and less cost is then required by DGTA to refine the result. The second observation is that, the effectiveness (i.e., density and entropy) of DSPA is the worst, followed by DAPA, DPPA, and DBA, while DGTA performs the best. The reason behind is that, DSPA trades the accuracy for efficiency, while DGTA can improve the quality due to the designed objective function. Finally, the density of our methods increases while the entropy drops (i.e., the quality improves) when δ grows. This is because, more compact partitions can be obtained when δ increases.

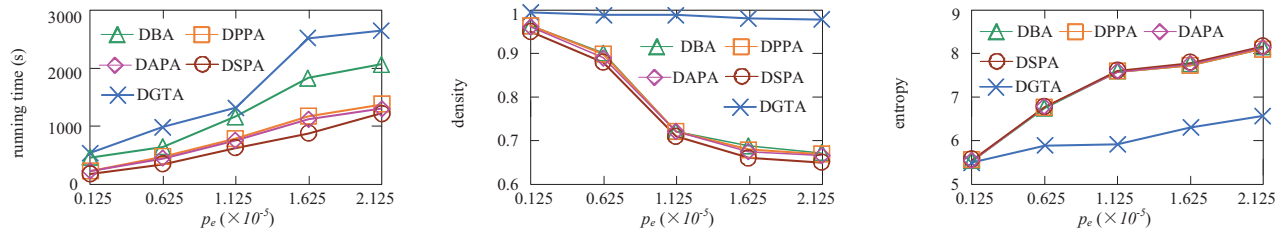
Second, we investigate the impact of $minPts$ on the efficiency and effectiveness of DBA, DPPA, DAPA, and DGTA. Fig. 5 shows the results when varying $minPts$ on *DBLP* and *Flickr*. As δ and $minPts$ are both used to define the density for DBSCAN, the observations of increasing the value of $minPts$ are similar to those of increasing the value of δ. However, the entropy first drops and then increases when $minPts$ ascends on *DBLP*, and the density first drops and then increases on *Flickr*. The reason is that, when $minPts$ grows, the clustering quality may drop due to more separated partitions, and the quality may increase due to more compact partitions. Hence, $minPts$ value that achieves the highest quality depends on the dataset distribution.

Third, we explore the impact of β on the performance of DGTA. Fig. 6 plots the results when changing β on *DBLP*. Note that, β only affects the performance of DGTA, as β controls the importance between AssignmentCost and StructuralCost in the game theory objective function. As observed, with the growth of β,



(a) DBLP (b) DBLP (c) DBLP

Fig. 9. Performance of Our Methods vs. c



(a) DBLP (b) DBLP (c) DBLP

Fig. 10. Performance of Our Methods vs. p_e

the running time and the entropy slightly drop in most cases. This is because, when β ascends, the importance of AssignmentCost increases that has influence on both density and entropy, while the importance of StructureCost decreases which has influence on density.

Next, we explore the impact of the sample rate r_s on the performance of DSPA. Fig. 7 shows the results when varying r_s on *DBLP*. Note that, only DSPA is influenced by r_s, which keeps consistency with discussions in Section 6.3. As expected, as r_s grows, the running time of DSPA increases, while the result quality (i.e., density and entropy) is improved. The reason behind is that, the larger r_s, the less messages are discarded by DSPA. Note that, when r_s reaches 1, the running time of DSPA is more than that of DPPA and DAPA. This is because, DSPA needs additional cost to do the sampling.

Then, we explore the impact of the number of worker nodes n_w on the performance of our methods on *DBLP*. Fig. 8 plots the results when changing n_w. It is observed that, as the number of worker nodes increases, the density and the entropy are stable while the running time drops. The reason is that, more worker nodes can provide stronger computing power, but can not improve the quality of clustering result. In addition, DBA and DGTA need much more computations (e.g., the PPR score computations and the objective function computations), and thus, their performance can be improved significantly with the growth of n_w.

In the sequel, we evaluate the impact of c on the performance of our methods. Fig. 9 illustrates the results by varying c from 0.1875 to 6 on *DBLP*. The first observation is that the quality of clustering result first improves and then drops with the growth of c. This is because the contribution of attribute similarity cannot be too small or too large. As observed, it achieves the best density performance when c = 3, and achieves the best entropy performance when c = 0.75. In addition, the running time increases as c grows. The reason behind is that the growth of c results in more information propagations between nodes when computing PPR scores.

Finally, we verify the influence of network topology on the performance of our methods. Here, we directly keep the hub and attribute nodes in *DBLP*, and generate the edges between nodes according to ER random network generation strategy, i.e., two nodes are connected with the probability p_e. Fig. 10 plots the results by varying p_e from 1.25×10^{-5} to 2.125×10^{-5}, where the corresponding number of generated edges are 1.3M, 1.9M, 3.03M, 3.77M, and 4.78M. Here, we fix $minPts$ to 3, while δ is set to 0.055, 0.05, 0.035, 0.023, and 0.022 accordingly. This is because, as the number of edges increases, the PPR score between two nodes decreases that requires smaller δ for DBSCAN. The first observation is that the clustering quality drops with the growth of p_e. This is because, as the node degree ascends, the probability that nodes in different clusters are connected also increases, resulting in worse clustering quality. In addition, the running time increases with the growth of p_e. The reason behind is that more connected nodes incur larger number of information propagations between nodes when computing PPR scores.

8 Conclusions

Star-schema heterogeneous graphs are a useful tool to represent and study a wide range of real-world attributed graphs, such as bibliographic information and social media graphs. Clustering such graphs can provide useful insights into underlying data and reality that the data concerns. With the rapidly growth of data volume, a distributed method is required. In view of this, we explore the problem of distributed clustering on star-schema heterogeneous graphs. We propose a Blogel based clustering framework, which uses a symmetric unified distance measure based on PPR, employs distributed DBSCAN for clustering, and updates the edge weights iteratively in order to balance the importance of different attribute types. Four methods are developed for computing PPR scores, i.e., distributed backward search algorithm (DBSA), distributed partial PPR algorithm (DPPA), distributed approximate PPR algorithm (DAPA), distributed sampling PPR algorithm (DSPA), where DPPA and DAPA improve efficiency by avoiding recomputing the PPR scores when edge weights are updated, while DSPA samples the messages between nodes to reduce the communication cost. In addition, distributed game theory based algorithm (DGTA) uses game theory to trade efficiency for high quality results. Extensive experiments using real-life datasets confirm the effectiveness and

efficiency of our presented methods. In the future, it is interest to further improve the efficiency of our framework. Also, the interactive result quality improvement is another promising direction.

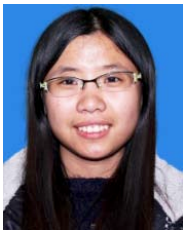
ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB1004003, the NSFC under Grant No. 62025206 and 61972338, and the NSFC Zhejiang Joint und under Grant No. U1609217. Yunjun Gao is the corresponding author of the work.

REFERENCES

[1] Y. Dodge, *Statistical data analysis based on the L1-norm and related methods*. Birkhäuser, 2012.
[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, 1996, pp. 226–231.
[3] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.
[4] B. Kulis, S. Basu, I. Dhillon, and R. Mooney, "Semi-supervised graph clustering: A kernel approach," *Machine learning*, vol. 74, no. 1, pp. 1–22, 2009.
[5] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, 026113, 2004.
[6] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: A structural clustering algorithm for networks," in *SIGKDD*, 2007, pp. 824–833.
[7] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: Applications to community discovery," in *SIGKDD*, 2009, pp. 737–746.
[8] G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "On clustering heterogeneous social media objects with outlier links," in *WSDM*, 2012, pp. 553–562.
[9] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip, "A survey of heterogeneous information network analysis," *TKDE*, vol. 29, no. 1, pp. 17–37, 2017.
[10] Y. Sun, C. C. Aggarwal, and J. Han, "Relation strength-aware clustering of heterogeneous information networks with incomplete attributes," *PVLDB*, vol. 5, no. 5, pp. 394–405, 2012.
[11] X. Li, Y. Wu, M. Ester, B. Kao, X. Wang, and Y. Zheng, "Semi-supervised clustering in attributed heterogeneous information networks," in *WWW*, 2017, pp. 1621–1629.
[12] A. Baroni, A. Conte, M. Patrignani, and S. Ruggieri, "Efficiently clustering very large attributed graphs," in *ASONAM*, 2017, pp. 369–376.
[13] H. Cheng, Y. Zhou, and J. X. Yu, "Clustering large attributed graphs: A balance between structural and attribute similarities," *TKDD*, vol. 5, no. 2, article no. 12, 2011.
[14] Y. Zhou, H. Cheng, and J. X. Yu, "Clustering large attributed graphs: An efficient incremental approach," in *ICDM*, 2010, pp. 689–698.
[15] L. Chen, Y. Gao, Z. Xing, C. S. Jensen, and G. Chen, "I2RS: A distributed geo-textual image retrieval and recommendation system," *PVLDB*, vol. 8, no. 12, pp. 1884–1887, 2015.
[16] H. Ma, J. Zhu, M. R.-T. Lyu, and I. King, "Bridging the semantic gap between image contents and tags," *IEEE Transactions on Multimedia*, vol. 12, no. 5, pp. 462–473, 2010.
[17] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci, "NG-DBSCAN: Scalable density-based clustering for arbitrary data," *PVLDB*, vol. 10, no. 3, pp. 157–168, 2016.
[18] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "A distributed and incremental algorithm for large-scale graph clustering," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 3, 2020.
[19] L. Chen, Y. Gao, Y. Zhang, C. S. Jensen, and B. Zheng, "Efficient and incremental clustering algorithms on star-schema heterogeneous graphs," in *ICDE*, 2019, pp. 256–267.
[20] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenkova, "Clustering attributed graphs: Models, measures and methods," *Network Science*, vol. 3, no. 3, pp. 408–444, 2015.
[21] J. D. Cruz, C. Bothorel, and F. Poulet, "Integrating heterogeneous information within a social network for detecting communities," in *ASONAM*, 2013, pp. 1453–1454.
[22] B. Boden, M. Ester, and T. Seidl, "Density-based subspace clustering in heterogeneous networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2014, pp. 149–164.
[23] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *SIGMOD*, 2012, pp. 505–516.
[24] C. Luo, W. Pang, and Z. Wang, "Semi-supervised clustering on heterogeneous information networks," in *PAKDD*, 2014, pp. 548–559.
[25] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu, "Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," *TKDD*, vol. 7, no. 3, article no. 11, 2013.
[26] B. Zheng, K. Zheng, P. Scheuermann, X. Zhou, Q. V. H. Nguyen, and C. Li, "Searching activity trajectory with keywords," *WWW Journal*, vol. 22, no. 3, pp. 967–1000, 2019.
[27] Y. Zhou and L. Liu, "Social influence based clustering and optimization over heterogeneous information networks," *TKDD*, vol. 10, no. 1, article no. 2, 2015.
[28] J. Chen, W. Dai, Y. Sun, and J. Dy, "Clustering and ranking in heterogeneous information networks via gamma-poisson model," in *SDM*, 2015, pp. 424–432.
[29] C. Shi, R. Wang, Y. Li, P. S. Yu, and B. Wu, "Ranking-based clustering on general heterogeneous information networks by network projection," in *CIKM*, 2014, pp. 699–708.
[30] Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu, "Rankclus: Integrating clustering with ranking for heterogeneous information network analysis," in *EDBT*, 2009, pp. 565–576.
[31] R. Wang, C. Shi, S. Y. Philip, and B. Wu, "Integrating clustering and ranking on hybrid heterogeneous information network," in *PAKDD*, 2013, pp. 583–594.
[32] M.-F. Balcan, S. Ehrlich, and Y. Liang, "Distributed clustering on graphs," *NIPS*, 2013.
[33] K. Hosseini, H. Dahrouj, and R. Adve, "Distributed clustering and interference management in two-tier networks," in *2012 IEEE Global Communications Conference*, 2012, pp. 4267–4272.
[34] O. Younis and S. Fahmy, "HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Transactions on mobile computing*, vol. 3, no. 4, pp. 366–379, 2004.
[35] L. Qing, Q. Zhu, and M. Wang, "Design of a distributed energy-efficient clustering algorithm for heterogeneous wireless sensor networks," *Computer communications*, vol. 29, no. 12, pp. 2230–2237, 2006.
[36] S. McClean, B. Scotney, K. Greer, and R. Pairceir, "Conceptual clustering of heterogeneous distributed databases," in *PKDD*, 2001, pp. 46–55.
[37] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[38] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *SIGMOD*, 2010, pp. 135–146.
[39] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *PVLDB*, vol. 7, no. 3, pp. 193–204, 2013.
[40] D. Yan, J. Cheng, Y. Lu, and W. Ng, "Blogel: A block-centric framework for distributed computation on real-world graphs," *PVLDB*, vol. 7, no. 14, pp. 1981–1992, 2014.
[41] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *PVLDB*, vol. 5, no. 8, pp. 716–727, 2012.
[42] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *SIGMOD*, 2013, pp. 505–516.
[43] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012, pp. 15–28.
[44] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *OSDI*, 2014, pp. 599–613.
[45] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. Mirrokni, and S.-H. Teng, "Local computation of pagerank contributions," *Algorithms and Models for the Web-Graph*, pp. 150–165, 2007.
[46] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka, "Efficient ad-hoc search for personalized pagerank," in *SIGMOD*, 2013, pp. 445–456.
[47] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," *PVLDB*, vol. 5, no. 5, pp. 442–453, 2012.
[48] T. Guo, X. Cao, G. Cong, J. Lu, and X. Lin, "Distributed algorithms on exact personalized pagerank," in *SIGMOD*, 2017, pp. 479–494.
[49] P. Lofgren, S. Banerjee, and A. Goel, "Personalized pagerank estimation and search: A bidirectional approach," in *WSDM*, 2016, pp. 163–172.

[50] S. Wang, Y. Tang, X. Xiao, Y. Yang, and Z. Li, "HubPPR: Effective indexing for approximate personalized pagerank," *PVLDB*, vol. 10, no. 3, pp. 205–216, 2016.
[51] H. Huang, Y. Gao, K. Chiew, L. Chen, and Q. He, "Towards effective and efficient mining of arbitrary shaped clusters," in *ICDE*, 2014, pp. 28–39.
[52] Z. Bu, H. J. Li, J. Cao, Z. Wang, and G. Gao, "Dynamic cluster formation game for attributed graph clustering," *IEEE Transactions on Cybernetics*, pp. 1–14, 2017.
[53] N. Armenatzoglou, H. Pham, V. Ntranos, D. Papadias, and C. Shahabi, "Real-time multi-criteria social graph partitioning: A game theoretic approach," in *SIGMOD*, 2015, pp. 1617–1628.
[54] H. Zhang, P. Lofgren, and A. Goel, "Approximate personalized pagerank on dynamic graphs," in *SIGKDD*, 2016, pp. 1315–1324.

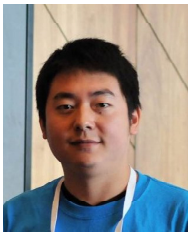


Lu Chen received the PhD degree in computer science from Zhejiang University, China, in 2016. She was an assistant professor in Aalborg University for a 2-year period from 2017 to 2019, and she was an associated professor in Aalborg University for a 1-year period from 2019 to 2020. She is currently a ZJU Plan 100 Professor in the College of Computer Science, Zhejiang University, Hangzhou, China. Her research interests include indexing and querying metric spaces, graph databases, and database usability.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor in the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, graph databases, spatio-textual data processing, and database usability. He is a member of the ACM and the IEEE.

Xingrui Huang is currently working toward the MS degree in College of Computer Science, Zhejiang University, China. His research interest includes graph databases.

Christian S. Jensen is an Obel professor of computer science with Aalborg University, Denmark. He was a professor with Aarhus University for a 3-year period from 2010 to 2013, and he was previously with Aalborg University for two decades. He recently spent a 1-year sabbatical with Google Inc., Mountain View, CA. His research concerns data management and data intensive systems, and its focus is on temporal and spatio-temporal data management. He is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several national and international awards for his research. He is editor-in-chief of the ACM Transactions on Database Systems and was an editor-in-chief of the VLDB Journal from 2008 to 2014. He is a fellow of the IEEE and ACM.



Bolong Zheng received the bachelor's and master's degrees in computer science from the Huazhong University of Science and Technology, in 2011 and 2013, respectively, and the PhD degree from the University of Queensland, in 2017. He is an associate professor with the Huazhong University of Science and Technology (HUST). His research interests include spatio-temporal data management and graph data management.