Efficient Structural Clustering on Probabilistic Graphs

Yu-Xuan Qiu, Rong-Hua Li, Jianxin Li, Shaojie Qiao, Guoren Wang, Jeffrey Xu Yu, and Rui Mao

Abstract—Structural clustering is a fundamental graph mining operator which is not only able to find densely-connected clusters, but it can also identify hub vertices and outliers in the graph. Previous structural clustering algorithms are tailored to deterministic graphs. Many real-world graphs, however, are not deterministic, but are probabilistic in nature because the existence of the edge is often inferred using a variety of statistical approaches. In this paper, we formulate the problem of structural clustering on probabilistic graphs, with the aim of finding reliable clusters in a given probabilistic graph. Unlike the traditional structural clustering problem, our problem relies mainly on a novel concept called reliable structural similarity which measures the probability of the similarity between two vertices in the probabilistic graph. We develop a dynamic programming algorithm with several powerful pruning strategies to efficiently compute the reliable structural similarities. With the reliable structural similarities, we adapt an existing solution framework to calculate the structural clustering on probabilistic graphs. Comprehensive experiments on five real-life datasets demonstrate the effectiveness and efficiency of the proposed approaches.

Index Terms—Probabilistic graph, structural clustering, reliable structural similarity

1 Introduction

Structural clustering has been recognized as an important graph mining operator. Different from traditional clustering, it not only discovers densely-connected clusters, but also is able to distinguish the different roles of the vertices in the graph by identifying the clustered vertices, hub vertices, and outliers [1], [2], [3], [4]. In the structural clustering framework (SCAN), the vertex included in any cluster is referred to as a clustered vertex; the vertex not included in any cluster is called a hub vertex if it connects no less than two clusters, and called an outlier otherwise [1], [4].

The previous structural clustering algorithm SCAN [1] and its improved implementations [3], [4] are tailored to deterministic networks. Many real-life networks, however, are probabilistic in nature, in which each edge is associated with a probability, representing the likelihood of the existence of the edge [5]. The probabilistic graphs have been widely used in many applications to model or express the inferred relationship of the vertices in a network [5]. Examples of such networks include protein-protein interaction networks with experimentally inferred links [6], sensor networks with uncertain connectivity links [7], social networks augmented with inferred friendship [8] or inferred influence [9]. Many fundamental data management and mining problems have recently been studied in the context of probabilistic graphs. Notable examples include the most reliable subgraph problem [10], the k-nearest neighbors search problem [11], the distance-constraint reachability computation problem [12], the frequent subgraph pattern mining problem [13], and the top-k maximal cliques problem [14].

The clustering problem on probabilistic graphs, with the aim of finding densely-connected subgraphs, has a number of applications in practice, two of which are listed below.

Application 1. Probabilistic graph clustering on uncertain traffic networks can help governments to identify the traffic hotspots in the city based on the local residents' transfer stops in their daily travel behaviors. Given two stops, residents may choose different routes to travel with different probabilities. Thus, probabilistic graph clustering can be used to discover the groups of stops that are highly reliable with regard to the residents' travel preferences. The areas bounded by such discovered stops would be the heavy traffic zones so that the governments may need to pay more attention when they plan some public events in such particular areas or zones.

Application 2. Probabilistic graph clustering can also be applied to study the cluster structures in protein-protein interaction (PPI) networks. In PPI networks, nodes denote proteins, and edges represent interactions among them. Each edge is associated with an uncertainty value because the interactions are derived through noisy and error-prone lab experiments. The edge of a protein pair indicates the reliability of the interaction between the two proteins [6]. Thus, finding the reliable clusters in PPI networks can discover the highly correlated proteins with probabilities at

- *Y.-X. Qiu and R.-H. Li are with the Beijing Institute of Technology, Beijing 100081, China, and the National Engineering Laboratory for Big Data System Computing Technology. E-mail: qiuyuxuan1@email.szu.edu.cn, lironghuascut@gmail.com.*
- *J. Li is with the University of Western Australia, Perth 6009, Australia. E-mail: jianxin.li@uwa.edu.au.*
- *S. Qiao is with the Chengdu University of Information Technology, Chengdu 610041, China. E-mail: sjqiao@cuit.edu.cn.*
- *G. Wang is with the Beijing Institute of Technology, Beijing 100081, China. E-mail: wanggrbit@126.com.*
- *J. X. Yu is with the Chinese University of Hong Kong, Hong Kong. E-mail: yu@se.cuhk.edu.hk.*
- *R. Mao is with Shenzhen University, Shenzhen 518060, China. E-mail: mao@szu.edu.cn.*

Manuscript received 23 Jan. 2018; revised 18 July 2018; accepted 22 Sept. 2018. Date of publication 28 Sept. 2018; date of current version 10 Sept. 2019. (Corresponding author: Rong-Hua Li.)
Recommended for acceptance by E. Terzi.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2872553

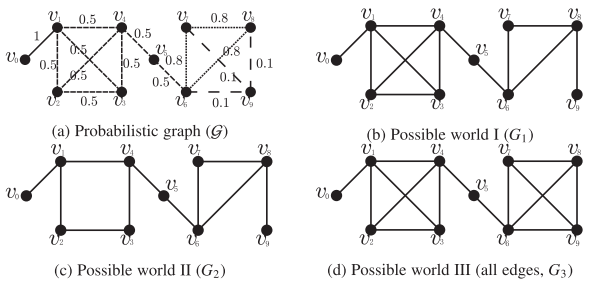


Fig. 1. Running example.

different levels, which can help to investigate the effect of medicines to the correlated proteins.

In the literature, researches on probabilistic graph clustering have also been studied. Kollios et al. [15] proposed a clustering algorithm by minimizing the expected edit distance between the probabilistic graph $\mathcal{G}$ and the cluster subgraph C. Each cluster subgraph C defined in this work requires to be a clique, and therefore their algorithm inevitably produces many small clusters. Liu et al. formulated a reliable clustering problem on probabilistic graphs and proposed a *coded k-means* algorithm to solve their problem[16]. The main deficiencies of their algorithm are twofold. First, it only works well for the applications that have a small number of ground-truth clusters [15]. Second, their algorithm cannot be scalable to large probabilistic graphs due to its quadratic space complexity.

All the existing clustering approaches focus mainly on partitioning the vertices into different clusters without recognizing their different roles in the probabilistic graph. It would make these methods limited in some real-life applications, e.g., understanding social users' behaviors and improving the advertisement placement in the influence network [9]. We could make significant improvements in these applications by distinguishing the clustered vertices, hubs, and outliers using a similar idea of structural clustering [1]. For example, the clustered vertices may be easily influenced by their neighborhood users, hub vertices might be influenced by their connected clusters, and outlier vertices may only be affected by their close neighbors. Motivated by this, in this paper we study the problem of *structural clustering on probabilistic graphs*, with the aim of identifying reliable clusters, hubs, and outliers in a given probabilistic graph.

To solve our problem, a straightforward method is to convert the probabilistic graph into a weighted deterministic graph, in which the weight of each edge represents the probability of it. Then, we invoke the traditional SCAN algorithm to derive the structural clustering results [1]. However, such a straightforward algorithm may fail to find reliable clusters. This is because the weighted deterministic graph may fail to reflect the *connectivity* of the probabilistic graphs correctly. We use the following example to illustrate this point.

Example 1. Consider a probabilistic graph $\mathcal{G}$ in Fig. 1a. When doing structural clustering on $\mathcal{G}$ without considering the probabilities on the edges, we may obtain two clusters, i.e., $\{v_1, v_2, v_3, v_4\}$ and $\{v_6, v_7, v_8, v_9\}$. This is equivalent to the situation of clustering the possible graph of $\mathcal{G}$ shown in Fig. 1d. However, there are many possible graphs of $\mathcal{G}$ like the graph shown in Fig. 1b or Fig. 1c, in which v_9 may not be in the same cluster with v_6, v_7 and v_8, because some of the edges between them may not be existed.

Contributions. In this paper, we formalize and provide efficient solutions to perform structural clustering on probabilistic graphs. In particular, we make the following contributions.

New concepts and problems. We first introduce a new concept called reliable structural similarity to measure the similarity between two vertices in the probabilistic graph. The proposed reliable structural similarity is able to capture the similarity between two vertices over all the possible instances of the probabilistic graph. Based on the reliable structural similarity, we formulate the structural clustering problem on probabilistic graphs, which is able to detect reliable clusters, hubs, and outliers in a given probabilistic graph.

New algorithms. To solve our problem, we first develop a polynomial-time dynamic programming (DP) algorithm to compute the reliable structural similarity exactly. Armed with the reliable structural similarities, we adapt the state-of-the-art SCAN algorithm to calculate the structural clustering results. We also develop several effective pruning techniques to further improve the efficiency of our algorithm.

Experimental evaluation. We conduct extensive experiments using five real-world datasets to evaluate the proposed algorithms. The results show that our algorithm significantly outperforms the baseline algorithms in terms of the clustering quality. The results also demonstrate the efficiency and scalability of the proposed algorithm. For example, our algorithm can obtain the structural clustering results on a million-sized probabilistic graph with 1,843,615 vertices and 8,350,259 edges in around one hour.

Organization. The rest of the paper is organized as follows. In Section 2, we introduce the problem formulation and some necessary background knowledge. In Section 3, we describe the clustering framework for the probabilistic graphs and a dynamic programming based algorithm to compute reliable structural similarity. We devise several optimization techniques to speed up our algorithms in Section 4. Section 5 presents the experimental results, and Section 6 reviews the related work. Finally, we conclude this work in Section 7.

2 Preliminaries

In this section, we first introduce some important notations and definitions, and then we formulate the structural clustering problem on probabilistic graphs.

2.1 Basic Notations and Definitions

Consider an unweighted and undirected probabilistic graph $\mathcal{G} = (V, E, P)$, where V is the set of vertices, E is the set of edges, and P denotes the set of probabilities. In $\mathcal{G}$, each edge $e \in E$ is associated with a probability $P_e \in P$. A probabilistic graph $\mathcal{G}$ can be instantiated into a deterministic graph by sampling each edge e with probability P_e. Following the standard uncertain graph model [10], [11], [17], [18], we assume that the existence of an edge is independent of that of any other edge. The well-known *possible world* semantics can be applied to analyze the probabilistic graphs [11]. Specifically, each possible world is a deterministic graph which contains all the vertices in V, but some or all of the edges in E, in terms of the probabilities in P.

Let $G = (V, E_G)$ be a possible world which is realized by sampling each edge in $\mathcal{G}$ according to the probability P_e. Clearly, we have $E_G \subseteq E$. The probability $Pr[G|\mathcal{G}]$ of sampling this possible world is calculated as follows.

$$Pr[G|\mathcal{G}] = \prod_{e \in E_G} P_e \prod_{e \in E \setminus E_G} (1 - P_e). \quad (1)$$

Example 2. Consider the probabilistic graph $\mathcal{G}$ shown in Fig. 1a. Figs. 1b, 1c, and 1d illustrate three possible worlds of $\mathcal{G}$, denoted by G_1, G_2, and G_3 respectively. Based on Eq. (1), we can easily derive that the probability of G_1, G_2, and G_3 are 0.000162, 0.000162, and 0.000002 respectively.

We make use of $G \sqsubseteq \mathcal{G}$ to indicate that G is a possible world of $\mathcal{G}$. Clearly, there are a total of $2^{|E|}$ possible worlds in graph $\mathcal{G}$ because each edge provides a binary sampling decision. For convenience, we use a notation $\mathcal{G}$ to denote a probabilistic graph, and utilize a notation G to denote a possible world or a deterministic graph. Below, we introduce some useful definitions of structural clustering for deterministic graph, which is originally proposed by Xu et al. [1].

Definition 1 (Structural Neighborhood [1]). *Given a deterministic graph $G = (V, E_G)$, the structural neighborhood of a vertex $u \in V$, denoted by $N[u]$, is the closed neighborhood of u, i.e., $N[u] = \{v \in V | (u, v) \in E_G\} \cup \{u\}$.*

It should be noted that the structural neighborhood of a vertex includes itself.

Example 3. Consider the graph in Fig. 1b, the structural neighborhood of vertex v_4 is $N[v_4] = \{v_1, v_2, v_3, v_4, v_5\}$. However, when considering Fig. 1c, we have $N[v_4] = \{v_1, v_3, v_4, v_5\}$.

Based on the definition of structural neighborhood, we give the definition of structural similarity.

Definition 2 (Structural Similarity). *Given a deterministic graph $G = (V, E_G)$, the structural similarity between vertices u and v, denoted by $\sigma(u, v)$, is defined as the number of common vertices in $N[u]$ and $N[v]$, normalized by $|N[u] \cup N[v]|$*

$$\sigma(u, v) = \frac{|N[u] \cap N[v]|}{|N[u] \cup N[v]|}. \quad (2)$$

Note that the original definition of structural similarity is based on a cosine-type similarity [1], here we adopt the Jaccard similarity, because it is more intuitive to measure the similarity between two vertices set. This Jaccard similarity was shown to be effective for structural clustering [2]. For brevity of the description, we also use $\sigma(e)$ to denote the structural similarity between u and v in the edge $e = (u, v)$.

Example 4. Consider the graph in Fig. 1d. Clearly, $N[v_1] = \{v_0, v_1, v_2, v_3, v_4\}$, $N[v_4] = \{v_1, v_2, v_3, v_4, v_5\}$, and $N[v_5] = \{v_4, v_5, v_6\}$. Thus, we have $|N[v_1] \cap N[v_4]| = |\{v_1, v_2, v_3, v_4\}| = 4$, $|N[v_1] \cup N[v_4]| = \{v_0, v_1, v_2, v_3, v_4, v_5\} = 6$, $|N[v_4] \cap N[v_5]| = |\{v_4, v_5\}| = 2$, and $|N[v_4] \cup N[v_5]| = \{v_1, v_2, v_3, v_4, v_5, v_6\} = 6$, and thereby we obtain that $\sigma(v_1, v_4) = \frac{|N[v_1] \cap N[v_4]|}{|N[v_1] \cup N[v_4]|} = \frac{4}{6} = \frac{2}{3}$, and $\sigma(v_4, v_5) = \frac{|N[v_4] \cap N[v_5]|}{|N[v_4] \cup N[v_5]|} = \frac{2}{6} = \frac{1}{3}$.

Definition 3 (ϵ-Structural Similarity [1]). *Given any two neighbor vertices u, v, and a similarity threshold ϵ, u is structural similar to v in a deterministic graph G if $\sigma(u, v) \geq \epsilon$ and $e = (u, v) \in E_G$.*

Example 5. Continuing the Example 4, if $\epsilon = 0.5$, v_1 is structural similar to v_4, as $\sigma(v_1, v_4) = 2/3 \geq \epsilon$. However, v_4 is not structural similar to v_5, because $\sigma(v_4, v_5) = 1/3 < \epsilon$.

2.2 Problem Formulation

The above definitions in the structural clustering algorithm (SCAN) are tailored to deterministic graphs. Below, we extend these definitions to probabilistic graphs.

Definition 4 (Probability of Structural Similarity). *Given a similarity threshold $0 < \epsilon \leq 1$, the probability of structural similarity that $\sigma(e) \geq \epsilon$ is defined as the sum of the probabilities of all the possible worlds $G \sqsubseteq \mathcal{G}$, such that the structural similarity of $e = (u, v)$ is no less than ϵ in each possible world G*

$$Pr[e, \epsilon] = \sum_{G \sqsubseteq \mathcal{G}} Pr[G|\mathcal{G}] \cdot \mathbf{I}(\sigma(e) \geq \epsilon), \quad (3)$$

where $I(\sigma(e) \geq \epsilon)$ is an indicator function which equals 1 if $\sigma(e) \geq \epsilon$, and 0 otherwise. If $e \notin E_G$, $I(\sigma(e) \geq \epsilon) = 0$.

Example 6. Consider the edge $e = (v_1, v_4)$ in the probabilistic graph $\mathcal{G}$ in Fig. 1a. Assume that $\epsilon = 0.5$. In the possible world G_1, $\sigma(e) = \frac{2}{3} \geq 0.5$, hence $I(\sigma(e) \geq 0.5) = 1$. Similarly, in G_2, we have $I(\sigma(e) \geq 0.5) = 0$, and $I(\sigma(e) \geq 0.5) = 1$ in G_3. By accumulating the probabilities of $I(\sigma(e) \geq \epsilon) = 1$ over all possible worlds of $\mathcal{G}$, we can derive that $Pr[e, 0.5] = 0.3125$.

On the basis of Definition 4, we define the reliable structural similarity between two vertices u and v in an edge $e = (u, v)$ as follows.

Definition 5 (Reliable Structural Similarity). *Given an edge $e = (u, v)$ and a threshold η, u is called reliable structural similar to v if $Pr[e, \epsilon] \geq \eta$.*

Clearly, by Definition 5, if u is reliable structural similar to v, u and v are structural similar with high probability. Intuitively, this definition captures the similarity between two vertices in the probabilistic graph. Moreover, as shown in Section 3, the striking feature of our reliable structural similarity based on Definitions 4 and 5 is that it can be computed in polynomial time, albeit the number of possible worlds in the probabilistic graph is exponentially large.

For a vertex u, the (ϵ, η)-reliable neighborhood of u contains all its neighbors that are reliable structural similar to u.

Definition 6 ((ϵ, η)-Reliable Neighborhood). *Given a similarity threshold $0 < \epsilon \leq 1$ and a probability threshold $0 < \eta \leq 1$, the (ϵ, η)-reliable neighborhood of u is defined as the subset of vertices in $N[u]$ such that $Pr[e = (u, v), \epsilon] \geq \eta$, i.e., $N_{(\epsilon,\eta)}[u] = \{v \in N[u] | Pr[e = (u, v), \epsilon] \geq \eta\}$.*

Example 7. Consider v_4 in the probabilistic graph $\mathcal{G}$ in Fig. 1a. Suppose that $\epsilon = 0.5$ and $\eta = 0.2$. Then, one can easily derive that $N_{(0.5,0.2)}[v_4] = \{v_1, v_2, v_3, v_4, v_5\}$ by definition.

The (ϵ, η)-reliable neighborhood is also called *reliable similar neighborhood*. Intuitively, the more reliable similar neighbors a vertex has, the more important role it plays in the clustering procedure. A vertex is termed as a reliable *core* vertex if it has a sufficient number of reliable similar neighbors.

Definition 7 ((ϵ, η, μ)-Reliable Core Vertex). *Given a similarity threshold $0 < \epsilon \leq 1$, a probability threshold $0 < \eta \leq 1$, and an integer $\mu \geq 2$, a vertex u is a (ϵ, η, μ)-reliable core vertex if $|N_{(\epsilon,\eta)}[u]| \geq \mu$.*

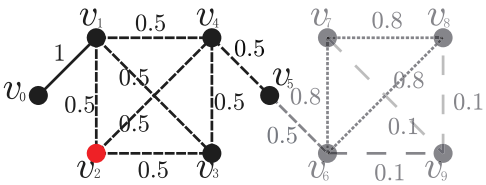


Fig. 2. Illustration of all the vertices that are reliable structure-reachable from v_2 ($\eta = 0.2, \epsilon = 0.5, \mu = 4$).

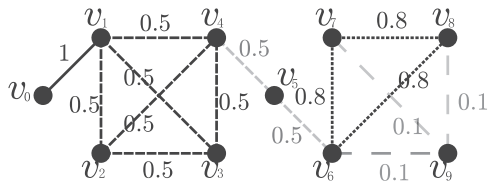


Fig. 3. Illustration of hubs and outliers ($C_1 = \{v_0, v_1, v_2, v_3, v_4\}$ and $C_2 = \{v_6, v_7, v_8\}$ are two clusters provided that $\eta = 0.2, \epsilon = 0.6, \mu = 3$).

Based on Definition 7, we give a definition of reliable structure-reachable as follows.

Definition 8 (Reliable Structure-reachable). *Given parameters $0 < \epsilon \leq 1$, $0 < \eta \leq 1$, and $\mu \geq 2$, vertex v is reliable structure-reachable from vertex u if there is a sequence of vertices $v_1, v_2, \ldots, v_l \in V$ ($l \geq 2$) such that:*

- *$v_1 = u$ and $v_l = v$;*
- *$v_1, v_2, \ldots, v_{l-1}$ are reliable core vertices;*
- *$v_{i+1} \in N_{\epsilon,\eta}(v_i)$ for each $1 \leq i \leq l-1$.*

In terms of Definition 8, if v is reliable structure-reachable from u, there is a path from u to v such that (i) u and all the intermediary vertices in the path are reliable core vertices, and (ii) the two vertices of each edge in the path are reliable structural similar. We use the following example to illustrate Definition 8.

Example 8. Reconsider the probabilistic graph in Fig. 1a. Let $\eta = 0.2$, $\epsilon = 0.5$, and $\mu = 4$. We can figure out that v_0, v_1, v_3, v_4 and v_5 are reliable structure-reachable from v_2. v_1, v_3, v_4 are reliable core vertices, and each of them is in $N_{(0.5,0.2)}[v_2]$. v_0 and v_5 are non-core vertices, $v_0 \in N_{(0.5,0.2)}[v_1]$, and $v_5 \in N_{(0.5,0.2)}[v_4]$. Note that v_6 is not reliable structure-reachable from v_2, albeit $v_6 \in N_{(0.5,0.2)}[v_5]$. This is because v_5 is a non-core vertex. Fig. 2 shows all the vertices that are reliable structure-reachable from v_2 (dark vertices).

Based on the above definitions, we formulate the structural clustering problem in probabilistic graphs as follows.

The Probabilistic Graph Clustering Problem. Given a probabilistic graph $\mathcal{G} = (V, E, P)$ and parameters $0 < \epsilon \leq 1$, $0 < \eta \leq 1$, and $\mu \geq 2$, the problem of probabilistic graph clustering is to compute the set $\mathbb{C}$ of *reliable clusters* in $\mathcal{G}$. Each *reliable cluster* $C \in \mathbb{C}$ should have at least two vertices (i.e., $|C| \geq 2$) and satisfy:

- Maximality: for each reliable core vertex $u \in C$, all vertices that are reliable structure-reachable from u must belong to C;
- Connectivity: for any two vertices $v_1, v_2 \in C$, there exists a vertex $u \in C$ such that both v_1 and v_2 are reliable structure-reachable from u.

Example 9. Consider the probabilistic graph in Fig 1a. Given that $\eta = 0.2, \epsilon = 0.5, \mu = 4$, we can obtain two clusters which are $C_1 = \{v_0, v_1, v_2, v_3, v_4, v_5\}$, $C_2 = \{v_5, v_6, v_7, v_8\}$ respectively. However, when $\eta = 0.2, \epsilon = 0.6, \mu = 3$, we are able to derive two different clusters which are $C_1 = \{v_0, v_1, v_2, v_3, v_4\}$ and $C_2 = \{v_6, v_7, v_8\}$ respectively.

Similar to the structural clustering results on deterministic graphs [4], the clusters obtained in our problem can also overlap. In Example 9, v_5 is assigned to both cluster C_1 and C_2. We can easily derive that each reliable core vertex can only be in one cluster. In addition, the proposed clustering procedure can also generate vertices that are not contained in any reliable clusters. These vertices are classified as hub vertices and outliers.

Definition 9 (Hub and Outlier). *Given the set $\mathbb{C}$ of reliable clusters in a probabilistic graph $\mathcal{G}$, a vertex u that is not in any reliable cluster in $\mathbb{C}$ is a* hub *vertex if it connects two or more reliable clusters, and it is an* outlier *vertex otherwise.*

Example 10. Consider the probabilistic graph $\mathcal{G}$ in Fig. 1a. Let $\eta = 0.2, \epsilon = 0.6, \mu = 3$, we are able to get two clusters which are $C_1 = \{v_0, v_1, v_2, v_3, v_4\}$ and $C_2 = \{v_6, v_7, v_8\}$, as illustrated in Fig. 3. Clearly, in this example, v_5 is a hub vertex, as it connects two clusters C_1 and C_2, whereas v_9 is an outlier, because it only links to a cluster C_2.

Given the set $\mathbb{C}$ of clusters in $\mathcal{G}$, it is straightforward to obtain the set of hubs and the set of outliers in the probabilistic graph $\mathcal{G}$ using $O(n + m)$ time by definition. In the following sections, we focus mainly on computing the set of clusters $\mathbb{C}$ in $\mathcal{G}$.

2.3 Challenges

Compared to the traditional structural clustering problem for deterministic graphs, our problem is much more complicated. This is because the structural similarity defined in our problem is associated with a probability (see Definition 4). Computing the probability of structural similarity is a challenging task, as it relies on a *sum of probabilities* over all possible worlds (see Definition 4). A straightforward method to compute this probability is to enumerate all possible worlds, and then determines whether the structural similarity is no less than the threshold ϵ in each possible world. This approach, however, is intractable for large probabilistic graphs, as the number of possible worlds is exponential. In the following section, we will develop an efficient dynamic programming algorithm to tackle this challenge, on the basis of an in-depth analysis of our problem. Note that after computing probability of structural similarity for each edge, we can easily adapt the existing SCAN algorithms to compute the reliable clusters, and identify hubs and outliers as well.

3 The Proposed Algorithm

In this section, we first introduce a clustering framework to solve our structural clustering problem on probabilistic graph by adapting the state-of-the-art PSCAN algorithm for structural clustering on deterministic graphs. Then, we develop a novel dynamic programming algorithm to compute the reliable structural similarities which is the most time-consuming step in our problem.

3.1 The Clustering Framework

The structural clustering framework for probabilistic graphs can be obtained by slightly modifying the state-of-the-art

PSCAN framework [4] for deterministic graphs. Note that PSCAN follows the same definition of structural clustering in deterministic graphs, which is originally proposed by Xu et al. [1]. The PSCAN framework is an improved solution with several optimizations to reduce the computational cost of the clustering procedure.

Algorithm 1. The USCAN Clustering Framework

Input: $\mathcal{G} = (V, E, P)$, and parameters ϵ, η, μ
Output: The set $\mathbb{C}$ of clusters in $\mathcal{G}$

1 Initialize $G_c = (V, \emptyset)$ such that each vertex in V is a connected component;
2 **for** *each vertex* $u \in V$ **do**
3 $c_u \leftarrow 0$; /* Initialize u as a non-core vertex */
4 **for** *each vertex* $u \in V$ **do**
5 **if** *IsReliableCore(u)* **then**
6 $c_u \leftarrow 1$;
7 **for** *each* $v \in N_{(\epsilon,\eta)}[u]$ **do**
8 **if** IsReliableCore(v) **then** Add (u, v) into G_c
9 $\mathbb{C}_c \leftarrow$ the set of connected components in G_c including reliable core vertices;
10 $\mathbb{C} \leftarrow \{C_c \cup_{u \in C_c} N_{(\epsilon,\eta)}[u] | C_c \in \mathbb{C}_c\}$;
11 **return** $\mathbb{C}$;
12 **Procedure** IsReliableCore(u)
13 $l_u \leftarrow 0$;
14 **for** *each vertex* $v \in N(u)$ **do**
15 Compute $Pr[e, \epsilon]$;
16 **if** $Pr[e, \epsilon] \geq \eta$ **then**
17 $l_u \leftarrow l_u + 1$;
18 **if** $l_u \geq \mu$ **then**
19 **Return** true;
20 **else**
21 **Return** false;

For completeness, we outline the modified PSCAN framework for probabilistic graphs in Algorithm 1 which is referred to as USCAN. Similar to PSCAN, the USCAN framework also consists of two stages. In the first stage, the algorithm clusters *reliable* core vertices, and then in the second stage, it clusters non-core vertices. Specifically, the algorithm first initializes every vertex as a cluster (line 1), and every vertex as a non-core vertex (lines 2-3). Then, for each vertex $u \in V$, the algorithm determines whether it is a reliable core vertex (lines 4-6). If u is a reliable core vertex, the algorithm visits each reliable neighbor of u. If such a reliable neighbor is also a reliable core vertex, the algorithm merges the clusters of u and v (lines 7-8). When this procedure terminates, the algorithm can obtain a set of connected components that contain the reliable core vertices (line 9). Finally, for each non-core vertex v, the algorithm puts it into the cluster containing a reliable core vertex u such that v is a reliable neighbor of u (line 10).

Note that the main difference between Algorithm 1 and the PSCAN algorithm proposed by Chang et al. is that our algorithm relies on computing the probability of structural similarity for each edge $e \in E$, while PSCAN is to compute traditional structural similarity [4]. We can also apply the pruning techniques developed in this work to speed up our clustering framework.

Correctness Analysis. The correctness of Algorithm 1 is guaranteed by the following lemmas which can be easily proven using similar arguments as shown by Chang et al. [4].

Lemma 1. *For any reliable cluster $C \in \mathbb{C}$ in a probabilistic graph $\mathcal{G}$, it comprises the set of vertices that are reliable structure-reachable form u, where u is an arbitrary reliable core vertex in C.*

Lemma 2. *For any two reliable core vertices u and v in $\mathcal{G}$, they must be in the same cluster in $\mathcal{G}$ if and only if they are in the same connected component in G_c.*

Lemma 3. *Let $\mathbb{C}_c$ be the set of clusters of reliable core vertices in $\mathcal{G}$, then the set of clusters of all vertices in $\mathcal{G}$ is $\{C_c \cup_{u \in C_c} N_{(\epsilon,\eta)}[u] | C_c \in \mathbb{C}_c\}$.*

Lemmas 1 and 2 indicate that all the connected reliable core vertices are contained in the same cluster. With Lemma 3, we are able to assign the neighbors of a reliable core vertex to their corresponding clusters. All the three lemmas make the maximality and connectivity to be satisfied in the probabilistic clustering problem.

3.2 Reliable Structural Similarity Computation

Recall that in Algorithm 1, we need to compute the (ϵ, η)-reliable neighborhood (i.e., $N_{(\epsilon,\eta)}[u]$), as well as determine whether a vertex is a reliable core. Both of these two operators rely mainly on calculating the probability of structural similarity (i.e., $Pr[e, \epsilon]$). As discussed in Section 2, computing $Pr[e, \epsilon]$ (the probability of $\sigma(e) \geq \epsilon$) is a very challenging task, because the number of possible instances of a probabilistic graph is exponential large. The straightforward algorithm to compute $Pr[e, \epsilon]$ based on Definition 4 has to explore all possible worlds, which is clearly intractable for large probabilistic graphs. Below, we devise a new dynamic programming approach to compute $Pr[e, \epsilon]$ in polynomial time based on a crucial observation of our definition.

DP Algorithm for $Pr[e, \epsilon]$ Computation. Let $\bar{G}$ be the deterministic counterpart of the probabilistic graph $\mathcal{G}$ (ignoring the probabilities of $\mathcal{G}$). Denote by $\bar{N}[u]$ the structural neighborhood of u in $\bar{G}$. Then, we have the following observation.

Observation 1. For each $e = (u, v) \in \mathcal{G}$, the number of possible values of the structural similarity between u and v over all the possible worlds can be bounded by $O(k_{join} \times k_{union})$, where $k_{join} = |\bar{N}[u] \cap \bar{N}[v]|$ and $k_{union} = |\bar{N}[u] \cup \bar{N}[v]|$.

Proof. Recall that in any possible world $G \sqsubseteq \mathcal{G}$, the structural similarity $\sigma(u, v)$ is equal to $|N[u] \cap N[v]|/|N[u] \cup N[v]|$, where $N[u]$ is the structural neighborhood of u in G. Clearly, $|N[u] \cap N[v]|$ must be an element in the integer set $\{0, 1, \ldots, k_{join}\}$, and $|N[u] \cup N[v]|$ must be a value in the integer set $\{0, 1, \ldots, k_{union}\}$. Therefore, the number of possible values of the structural similarity between u and v is no larger than $(k_{join} + 1) \times (k_{union} + 1)$. □

Example 11. Consider an edge (v_4, v_5) in the probabilistic graph in Fig. 1a. Since $k_{join} = |N[v_4] \cap N[v_5]| = 2$ and $k_{union} = |N[v_4] \cup N[v_5]| = 6$, $\sigma(v_4, v_5)$ must be a value in the set $S = \{0, 1, 1/2, 1/3, 1/4, 1/5, 1/6, 2/3, 2/5\}$, where $|S| = 9 < (k_{join} + 1) \times (k_{union} + 1) = 21$.

By Observation 1, we can see that the number of possible values of the structural similarity is bounded, albeit there are $2^{|E|}$ possible worlds in $\mathcal{G}$. This result suggests that for an edge (u, v), we can group all possible worlds into a class if the $\sigma(u, v)$ values on these possible worlds are equal to the same value. Following this way, we can obtain at most $O(k_{join} \times k_{union})$ classes, where each possible structural

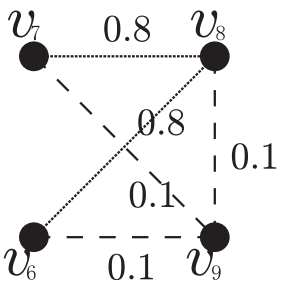


Fig. 4. Running example of calculating $Pr[(v_8, v_9), \epsilon]$.

similarity value represents a class. By enumerating all these classes (instead of enumerating all possible worlds), we are able to obtain a polynomial-time algorithm to compute the probability of structural similarity. Below, we show how to establish a recursive relationship between different classes.

Let $e = (u, v)$ be a probabilistic edge in $\mathcal{G}$, $N(u) = N[u] \setminus \{u\}$ and $N(v) = N[v] \setminus \{v\}$, denoting the number of possible neighbors of u and v, respectively. Assume, without loss of generality, that the vertices in $N(u) \cup N(v)$ are ordered by their vertices IDs. Then, we process one vertex in $N(u) \cup N(v)$ at a time in the probabilistic graph, following the vertex order. Suppose that we have already processed h vertices by the hth time and w is the vertex to be processed at the $(h+1)$th time. Consider the case when $\sigma(u, v) = \frac{m'}{n'}$, where u and v have m' common neighbors, and they have n' neighbors in total. Then, the probability of $\sigma(u, v) = \frac{m'}{n'}$ at the $(h+1)$th time, denoted by $Pr[\sigma(e) \overset{h+1}{=} \frac{m'}{n'}]$, can be calculated by accessing the computational results at the previous h times.

Let $e_1 = (u, w)$ and $e_2 = (v, w)$ be two potential edges. Then, $\sigma(e) \overset{h+1}{=} \frac{m'}{n'}$ holds if and only if one of the following three cases happens:

(i): Both e_1 and e_2 exist in the possible worlds where e occurs. In this case, the probability is equal to $P_{e_1} P_{e_2} Pr[\sigma(e) \overset{h}{=} \frac{m'-1}{n'-1}]$, which requires to aggregate the probabilities of the cases that have $m' - 1$ common neighbors among the total $n' - 1$ neighbors processed at the previous h times.

(ii): Only one of e_1 and e_2 exists in the possible worlds where e occurs. The probability in this case is $((1 - P_{e_1})P_{e_2} + P_{e_1}(1 - P_{e_2}))Pr[\sigma(e) \overset{h}{=} \frac{m'}{n'-1}]$, which requires to aggregate the probabilities of the cases that have m' common neighbors among the total $n' - 1$ neighbors processed at the previous h times.

(iii): Neither e_1 nor e_2 exists in the possible worlds where e occurs. The probability is $(1 - P_{e_1})(1 - P_{e_2})Pr[\sigma(e) \overset{h}{=} \frac{m'}{n'}]$, which needs to aggregate the probabilities of the cases that have m' common neighbors among the total n' neighbors processed at the previous h times.

Based on the above three cases, we can derive that the probability of structural similarity of e at the $(h+1)$th time can be computed in a recursive way as follows.

$$\begin{aligned} Pr\left[\sigma(e) \overset{h+1}{=} \frac{m'}{n'}\right] &= P_{e_1} P_{e_2} Pr\left[\sigma(e) \overset{h}{=} \frac{m'-1}{n'-1}\right] \\ &+ ((1 - P_{e_1})P_{e_2} + P_{e_1}(1 - P_{e_2}))Pr\left[\sigma(e) \overset{h}{=} \frac{m'}{n'-1}\right] \\ &+ (1 - P_{e_1})(1 - P_{e_2})Pr\left[\sigma(e) \overset{h}{=} \frac{m'}{n'}\right]. \end{aligned} \quad (4)$$

By Eq. (4), we can devise a DP algorithm to compute $Pr[\sigma(e) \overset{h+1}{=} \frac{m'}{n'}]$ for all h, m' and n'. Denote by $W = N(u) \cup N(v) = \{w_1, w_2, \ldots, w_{k_{union}-2}\}$. At the hth time, the algorithm has processed h vertices, i.e., $W_h = \{w_1, \ldots, w_h\} \subseteq W$, where $1 \leq h \leq k_{union}$. Let $X(h, m', n')$ be a state, representing the probability of $\sigma(e) = \frac{m'}{n'}$ after processing the vertex w_h at the h-time. Then, based on Eq. (4), the state-transformation equation of the DP algorithm is given as follows.

$$\begin{aligned} X(h, m', n') &= p_{(w_h,u)} p_{(w_h,v)} X(h-1, m'-1, n'-1) \\ &+ ((1 - p_{(w_h,u)})p_{(w_h,v)} + p_{(w_h,u)}(1 - p_{(w_h,v)}))X(h-1, m', n'-1) \\ &+ ((1 - p_{(w_h,u)})(1 - p_{(w_h,v)}))X(h-1, m', n'). \end{aligned} \quad (5)$$

Initially, the basic state of our DP algorithm can be set as $X(0, 2, 2) = p_e$ for an edge e. This is because $m' = 2$ and $n' = 2$ when no vertex has been processed, and the probability of this case is equal to the probability p_e.

Algorithm 2 outlines the detailed implementation of our DP algorithm. The algorithm first initializes $X(h, m', n') = 0$ and $Pr(e, \epsilon) = 0$ (lines 1-2). Then, the algorithm sets $X(0, 2, 2) = 1$ (line 3). To ensure the correctness, the algorithm will multiply p_e when it terminates (line 11). Subsequently, in lines 4-7, the algorithm continuously processes vertices and enumerates all possible situations of the numerators and denominators of the structural similarity based on the Eq. (5). In lines 8-10, the algorithm computes the sum of all $X(h, m', n')$ for $\frac{m'}{n'} \geq \epsilon$, resulting in $\Pr(e, \epsilon)$. Finally, the algorithm returns $p_e \times \Pr(e, \epsilon)$ as the result. The correctness of Algorithm 2 can be guaranteed by Eqs. (4) and (5). The following example illustrates how Algorithm 2 works.

Example 12. Considering the edge $e = (v_8, v_9)$ in Fig. 4 which is a subgraph of Fig. 1a. Assume that $\epsilon = 0.6$, and we aim to calculate $Pr[e, 0.6]$. First, the algorithm initializes $X(0, 2, 2) = 1$. Then, the algorithm processes the vertex v_6, and enumerates all potential m' and n' based on Eq. (5). Specifically, in this iteration, the algorithm computes $X(1, 2, 2) = X(0, 2, 2) \times (1 - 0.1) \times (1 - 0.8) = 0.18$, $X(1, 2, 3) = X(0, 2, 2) \times ((1 - 0.1) \times 0.8 + 0.1 \times (1 - 0.8)) = 0.74$, $X(1, 3, 3) = 0.08$. Subsequently, the algorithm processes the vertex v_7, and calculates $X(2, 2, 2) = X(1, 2, 2) \times (1 - 0.1) \times (1 - 0.8) = 0.0324$, $X(2, 2, 3) = 0.2664$, $X(2, 3, 3) = 0.0288$, $X(2, 2, 4) = 0.5476$, $X(2, 3, 4) = 0.1184$, $X(2, 4, 4) = 0.0064$. After obtaining all the probabilities for each possible m' and n', the algorithms takes the sum over all probabilities of $\frac{m'}{n'} \geq 0.6$, and then multiplies P_e to derive that $Pr[e, 0.6] = (0.0324 + 0.2664 + 0.0288 + 0.1184 + 0.0064) \times 0.1 = 0.04524$.

Complexity Analysis. First, it is easy to show that the time complexity of Algorithm 2 to compute $Pr(e, \epsilon)$ for an edge $e = (u, v)$ is $O(k_{union}^2 k_{join})$, where $k_{join} = |\bar{N}[u] \cap \bar{N}[v]|$ and $k_{union} = |\bar{N}[u] \cup \bar{N}[v]|$. Note that k_{union} can be bounded by $2 \times d_{\max}$, where $d_{\max}$ is the maximum degree in $\bar{G}$ (the deterministic counterpart of the probabilistic graph $\mathcal{G}$), and k_{join} is bounded by $\min\{d_u, d_v\}$. Thus, the time cost of Algorithm 2 to process an edge (u, v) is bounded by $O(d_{\max}^2 \times \min\{d_u, d_v\})$. As a consequence, the total cost of Algorithm 2 to compute $Pr(e, \epsilon)$ for all $e \in E$ can be bounded by $O(d_{\max}^2 \times \sum_{(u,v)\in E} \min\{d_u, d_v\}) = O(d_{\max}^2 \times \alpha \times m)$, where α denotes the arboricity of the graph $\mathcal{G}$ and $m = |E|$. Note that α can be bounded by $O(\sqrt{m})$ [19], and it is typically much

smaller than the worst-case bound in real-world graphs [20]. Clearly, after computing the reliable structural similarities for all edges, the entire clustering procedure can be done in linear time (with respect to the graph size). As a result, the total time cost of our algorithm for structural clustering on probabilistic graphs is $O(d_{\max}^2 \times \alpha \times m)$.

Algorithm 2. DP for Computing $\Pr(e, \epsilon)$

Input: $\mathcal{G} = (V, E, P)$, an edge $e = (u, v) \in E$, and similarity threshold ϵ
Output: the probability $Pr(e, \epsilon)$ when the structural similarity of e is no less than ϵ

1 Initialize $X(h, m', n') \leftarrow 0$, for all $h \in [0, k_{union}]$, $m' \in [0, k_{join}]$ and $n \in [0, k'_{union}]$;
2 $Pr(e, \epsilon) \leftarrow 0$;
3 $X(0, 2, 2) \leftarrow 1$;
4 **for** $h \leftarrow 1$ **to** $k_{union} - 2$ **do**
5 **for** $n' \leftarrow 2$ **to** k_{union} **do**
6 **for** $m' \leftarrow 2$ **to** $\min\{n', k_{join}\}$ **do**
7 $X(h, m, n) \leftarrow p_{(w_h,u)} p_{(w_h,v)} X(h-1, m'-1, n'-1) + ((1 - p_{(w_h,u)}) p_{(w_h,v)} + p_{(w_h,u)}(1 - p_{(w_h,v)})) X(h-1, m', n'-1) + ((1 - p_{(w_h,u)})(1 - p_{(w_h,v)})) X(h-1, m', n')$
8 **for** $n' \leftarrow 2$ **to** k_{union} **do**
9 **for** $m' \leftarrow \lceil n'\epsilon \rceil$ **to** $\min\{n', k_{join}\}$ **do**
10 $Pr(e, \epsilon) \leftarrow Pr(e, \epsilon) + X(k_{union} - 2, m', n')$
11 **return** $P_e * Pr(e, \epsilon)$

For the space complexity, Algorithm 2 only needs to maintain two 2-dimensional (2D) arrays to compute all $X(h, m', n')$ for each edge (u, v), which consumes $O(k_{union} k_{join})$. This is because the states in the hth iteration only rely on the states in the $(h-1)$th iteration (see Eq. (5)), thus two 2D arrays are sufficient to implement the DP algorithm. Since our structural clustering algorithm can release the space after computing $\Pr(e, \epsilon)$ for each edge e, the space cost of our algorithm can be bounded by $O(d_{\max}^2 + m)$. In practice, the space usage of our algorithm is much less than the worst-case bound, because $k_{union} k_{join}$ is typically much smaller than $d_{\max}^2$ for most edges. In our experiments, we will show that the space overhead of our algorithm is slightly larger than the graph size.

4 Optimization

Recall that in our algorithm, the most time-consuming step is to compute the probability of structural similarity for each edge, i.e., $Pr[e, \epsilon]$ for each $e \in E$. In this section, we develop several effective pruning techniques to speed up the computing of $Pr[e, \epsilon]$. Below, we first introduce the basic pruning rules, followed by the early termination pruning rule, as well as several advanced pruning rules.

4.1 Basic Pruning Rules

Property 1 (Pruning Improper Edges). *For any edge $e = (u, v) \in E$, if $P_e < \eta$, we have $Pr[e, \epsilon] < \eta$.*

When the probability P_e of an edge e is less than η, the probability that e exists in all possible worlds is also less than η. Hence, it is easy to derive that the probability $Pr[e, \epsilon]$ of the structural similarity on e must be less than η. As a consequence, we can prune all the edges with probabilities less than η in $\mathcal{G}$ using Property 1.

Property 2 (Avoiding Duplicate Computation). *For any edge $e = (u, v) \in E$, $Pr[e = (u, v), \epsilon] = Pr[e = (v, u), \epsilon]$ always holds.*

Consider that $Pr[e = (u, v), \epsilon]$ is computed when we visit the vertex u. Based on the Property 2, it is equivalent to have $Pr[(v, u), \epsilon]$ for the unvisited vertex v. By doing this, we can avoid the duplicate computation on the edge $e = (v, u)$ when we visit v.

Recall that at the hth iteration, Algorithm 2 needs to enumerate each possible value of n' (the second "for" loop), and then enumerates each possible value of m' (the third "for" loop). Note that for any state $X(h, n', m')$ in the DP procedure, n' must be no less m', because n' denotes the denominator of the Jaccard similarity, while m' represents the numerator of the Jaccard similarity. As a result, we are capable of using the property of $m' \leq n'$ for pruning. Specifically, in the third loop, we only need to enumerate m' from 2 to $\min\{n', k_{join}\}$ which is implemented in Algorithm 2 (Line 6).

4.2 Early Termination

Intuitively, we may reduce the computational cost of the DP algorithm by checking the condition that $\frac{m'}{n'} \geq \epsilon$, i.e., $m' \geq n' \times \epsilon$. However, it is nontrivial to use this result for pruning. Specifically, the challenge is that even if $\frac{m'}{n'} < \epsilon$ in the third loop of Algorithm 2, we are still not able to terminate the computation, otherwise we may obtain incorrect results. This is because the value of $\frac{m'}{n'}$ may increase to $\frac{m'+1}{n'+1}$ if the algorithm continues to enumerate the next common neighbor, where $\frac{m'+1}{n'+1}$ could be no less than ϵ; and therefore we cannot terminate the algorithm when $\frac{m'}{n'} < \epsilon$. Below, we present an approach to tackle this challenge.

Our approach is based on the following property.

Property 3. *Suppose that all the common neighbors in $N(u) \cap N(v)$ have been processed by the DP algorithm at the hth iteration. We can terminate the computation to enumerate the values of n' or m' if $\frac{m'}{n'} < \epsilon$ at the hth iteration.*

Proof. Since all the common neighbors in $N(u) \cap N(v)$ have been processed at the hth iteration, the value of $\frac{m'}{n'}$ cannot increase after enumerating any neighbor in $N(u) \cup N(v)$ (because enumerating any neighbor in $N(u) \cup N(v)$ can only increase the denominator of the Jaccard similarity). Therefore, if we have $\frac{m'}{n'} < \epsilon$ at the hth iteration, the algorithm can early terminate the enumeration of n' or m'. □

We implement this early terminating pruning in Algorithm 3 (Line 5-8). Note that in Algorithm 3, we assume that all the common neighbors are processed first, and then the algorithm process the other neighbors in $N(u) \cup N(v)$. Thus, in Line 8, we can use the condition $h > k_{join} - 2$ to determine whether all common neighbors have been explored. Unlike Algorithm 2, Algorithm 3 first enumerates m', and then enumerates n' to further reduce the computational cost. This is because n' is typically much larger than m', and therefore the algorithm can significantly reduce the computational cost for enumerating n' using the early termination pruning rule.

4.3 Pruning by Lower and Upper Bounds

Here we develop several lower and upper bounds for $Pr[e, \epsilon]$, which can be used to further reduce the computational cost of Algorithm 3. Specifically, if we find that the lower bound of $Pr[e = (u, v), \epsilon]$ is larger than η, we can early complete the algorithm, because in this case u must be reliable structural similar to v. Likewise, if we derive that the upper bound of $Pr[e = (u, v), \epsilon]$ is smaller than η, we are

also able to terminate the algorithm, as u is definitely not reliable structural similar to v in this case. We details our lower and upper bounding techniques as follows.

Algorithm 3. Improved DP for Computing $\Pr(e, \epsilon)$

```
1  if p_e < η then
2    return; /* Property 1 pruning rule */
3  Lines 1-3 in Algorithm 2;
4  for h ← 1 to k_union − 2 do
5    for m' ← 2 to min{h + 2, k_join} do
6      τ ← m'/ε;
7      for n' ← m' to min{h + 2, k_union} do
8        if h > k_join − 2 and n' ≥ τ then
9          break; /* early termination */
10       X(h, m', n') ← p_(w_h,u) p_(w_h,v) X(h − 1, m' − 1, n' − 1) +
         ((1 − p_(w_h,u)) p_(w_h,v) + p_(w_h,u)(1 − p_(w_h,v))) X(h − 1, m',
         n' − 1) + ((1 − p_(w_h,u))(1 − p_(w_h,v))) X(h − 1, m', n')
11 Lines 8-11 in Algorithm 2;
```

Fixed Denominator Based Lower Bound. Recall that in any possible world, the denominator of the Jaccard similarity between u and v ($\sigma(u, v)$) is no larger than k_{union}, i.e., $n' \leq k_{union}$. In any possible world, if we fix the denominator as k_{union} in computing $\sigma(u, v)$, the result must be no larger than the exact $\sigma(u, v)$. Therefore, we can derive a lower bound for $Pr[e, \epsilon]$ by computing $\sigma(u, v)$ with a fixed denominator k_{union} in any possible world.

We can devise a lightweight DP algorithm to compute such a lower bound. Similar to Algorithm 2, we denote $X(h, m')$ as a state representing the probability of $\sigma(u, v) = m'/k_{union}$ after processing the vertex w_h at the hth iteration, where $w_h \in N(u) \cup N(v)$. Then, the following recursive equation can be applied to compute all $X(h, m')$.

$$\begin{aligned} X(h, m') &= p(w_h, u)p(w_h, v)X(h - 1, m' - 1) \\ &+ ((1 - p(w_h, u))p(w_h, v) + p(w_h, u)(1 - p(w_h, v)) \\ &+ (1 - p(w_h, u))(1 - p(w_h, v)))X(h - 1, m'). \end{aligned} \quad (6)$$

Initially, we have $X(0, 2) = p_e$. The DP algorithm can be easily devised based on Eq. (6), we omit the details due to the space limit. Obviously, the time complexity of this lightweight DP algorithm is $O(k_{union} \times k_{join})$, which is much lower than that of the Algorithm 2. The space complexity of this DP algorithm is $O(k_{join})$ using a similar space-saving trick as presented in Algorithm 2. After computing all $X(h, m')$ for $h \in [0, k_{union}]$ and $m' \in [0, k_{join}]$, we are able to obtain a lower bound $\sum_{\forall h, m', m'/k_{union} \geq \epsilon} X(h, m')$ for $Pr[e, \epsilon]$.

Fixed Numerator Based Upper Bound. Similarly, an upper bound for $Pr[e, \epsilon]$ can be obtained by fixing the numerator m' as k_{join} for the Jaccard similarity $\sigma(u, v)$ in any possible world. This is because, by fixing $m' = k_{join}$, the resulting Jaccard similarity between u and v in any possible world is k_{join}/n' which is no smaller than the exact Jaccard similarity.

Likewise, we can develop a similar lightweight DP algorithm to compute the upper bound. Let $X(h, n')$ be the state, representing the probability of $\sigma(u, v) = k_{join}/n'$ after processing the vertex w_h at the hth iteration. Then, the recursive equation of this lightweight DP algorithm is shown as follows.

$$\begin{aligned} X(h, n') &= (p(w_h, u)p(w_h, v) + (1 - p(w_h, u))p(w_h, v) \\ &+ p(w_h, u)(1 - p(w_h, v)))X(h - 1, n' - 1) \\ &+ ((1 - p(w_h, u))(1 - p(w_h, v)))X(h - 1, n'). \end{aligned} \quad (7)$$

Clearly, the time and space complexities of this lightweight DP algorithm are $O(k_{union})^2$ and $O(k_{union})$ respectively, which are much lower than those of Algorithm 2. Similarly, we can obtain an upper bound $\sum_{\forall h, n', k_{join}/n' \geq \epsilon} X(h, n')$ for $Pr[e, \epsilon]$, after computing all $X(h, n')$ for $h \in [0, k_{union}]$ and $n' \in [0, k_{union}]$.

A Tight Upper Bound. Here we derive a novel upper bound which is tighter than the fixed numerator based upper bound. For any edge $e = (u, v)$, we first remove all the neighbor vertices in $(N(u) \cup N(v)) \setminus (N(u) \cap N(v))$. Then, we compute the probability of structural similarity on the revised probabilistic graph only containing the common neighbors between u and v. The computed probability, denoted by $\tilde{Pr}[e, \epsilon]$, is an upper bound of $Pr[e, \epsilon]$. The reason is that adding back any vertex in $(N(u) \cup N(v)) \setminus (N(u) \cap N(v))$ into the revised probabilistic graph may increase the denominator of the Jaccard similarity $\sigma(e)$, and therefore decreases the probability of "$\sigma(e) \geq \epsilon$". Hence, the exact probability $Pr[e, \epsilon]$ must be no larger than $\tilde{Pr}[e, \epsilon]$.

Interestingly, we can slightly modify Algorithm 3 to compute $\tilde{Pr}[e, \epsilon]$. Similar to Algorithm 3, the modified algorithm processes the common neighbors between u and v first, and then processes the other neighbors. Once the algorithm has processed all the common neighbors, we can easily derive that $\tilde{Pr}[e, \epsilon] = \sum_{h \leq |N(u) \cap N(v)|, \forall m', n', m'/n' \geq \epsilon} X(h, m', n')$. This is because, when $h \leq |N(u) \cap N(v)|$, the DP algorithm only examines the common neighbors which is equivalent to performing the DP procedure on the revised probabilistic graph that only contains the common neighbors between u and v. The modified algorithm only needs to perform an additional computing of $\sum_{h \leq |N(u) \cap N(v)|, \forall m', n', m'/n' \geq \epsilon} X(h, m', n')$, when it has processed all common neighbors. Note that if $\tilde{Pr}[e, \epsilon] < \eta$, we can early terminate the DP algorithm, because $Pr[e, \epsilon]$ is definitely smaller than η. In addition, $\tilde{Pr}[e, \epsilon]$ is tighter than the fixed numerator based upper bound. This is because the numerator of the Jaccard similarity in computing $\tilde{Pr}[e, \epsilon]$ is definitely no larger than k_{join}, thus the probability of the structural similarity with fixed numerator k_{join} must no smaller than $\tilde{Pr}[e, \epsilon]$.

5 Experiments

In this section, we conduct extensive experiments to evaluate the efficiency and effectiveness of the proposed algorithms. For our algorithms, we implement three variants: USCAN-B, USCAN-E, USCAN-A. USCAN-B is essentially the USCAN algorithm with the basic pruning techniques, USCAN-E is the USCAN-B algorithm with the early termination pruning, and USCAN-A is our USCAN algorithm with all pruning techniques proposed in Section 4. To evaluate the effectiveness of our algorithms, we implement two state-of-the-art uncertain graph clustering algorithms: PCluster and CKmeans. PCluster is the probabilistic graph clustering algorithm proposed by Kollios et al. [15], while CKmeans denotes the reliable graph clustering algorithm presented by Liu et al. [16]. In addition, we implement a sampling based SCAN algorithm called MSampling as a baseline. In MSampling, we first sample N possible worlds, and then calculate the expected structural similarity of the edges. With these similarities, we invoke the SCAN algorithm for clustering. All algorithms are implemented in C++ and the Standard Template Library (STL) is used. All

TABLE 1
Datasets with Detailed Statistics (d_{max}: Max Degree, $\bar{d}$: Average Degree, and $\bar{p}$: Average Probability)

Graph	#vertices	#edges	d_{max}	$\bar{d}$	$\bar{p}$
CORE	2,708	7,123	141	5.26	0.6794
DBLP01	435,646	1,005,796	241	4.62	0.4736
DBLPAll	1,843,615	8,350,259	2,213	9.06	0.4926
Amazon	334,863	925,872	549	5.53	0.5001
Youtube	1,134,890	2,987,624	28,754	5.27	0.5001

TABLE 2
Clustering Precision of Various Algorithms

Algorithm	#clusters	TP	FP	PR
Reference	547	1,765	11,692	0.131
USCAN-A	456	1,086	2,037	**0.348**
MSampling	384	1,331	3,223	0.292
PCluster	475	1,027	3,021	0.266
CKmeans	150	550	899,145	0.00061

the experiments are conducted on an Ubuntu 16.04.1 Linux System with Intel(R) Xeon(R) CPU E5-2620 v3 (2.40 GHz) and 32GB main memory.

Datasets. We use five various real-life datasets in the experiments. The detailed information of these datasets are described as follows. CORE is a protein-protein interaction network provided by Krogan et al. [21]. The network contains 2,708 vertices and 7,123 edges, in which a vertex denotes a protein and an edge represents an interaction between two proteins. In this network, each edge is associated with a probability, denoting the confidence of the interaction between two proteins. There are around 20 percent edges that have a probability no less than 0.98, and no edge with probability less than 0.27. The edge probabilities are uniformly distributed in the range $[0.27, 0.98]$. The dataset shows power-law degree distribution, small diameter, and high cluster coefficient. DBLP is a co-authorship network. We download the original XML data from (http://dblp.uni-trier.de/xml) to extract the co-authorship network, where two authors are connected with a link if they co-author at least one paper. We collect two different DBLP datasets to test our algorithms: DBLP01 and DBLPAll. DBLP01 consists of a co-authorship network before 2001, whereas DBLPAll comprises the entire co-authorship network up to date. We mainly use DBLP01 for general testings, because the baseline algorithm CKmeans cannot handle large graphs (e.g., DBLPAll). For DBLPAll, we will use it to evaluate the scalability of our algorithms. For these two DBLP datasets, we can obtain a weight for each edge which is the number of papers co-authored by two researchers. To generate a probabilistic graph for DBLP01 and DBLPAll, we adopt a standard method in the uncertain graph mining literature [11], [12]. In particular, for each edge (u, v), we make use of an exponential cumulative distribution with mean 2 to the weight of (u, v) to generate a probability. Amazon and Youtube are two unweighted social networks downloaded from the Stanford Network Analysis Platform (http://snap.stanford.edu/). For these two unweighted networks, we generate a probability for each edge following a uniform distribution. The detailed statistic information of all our datasets are reported in Table 1.

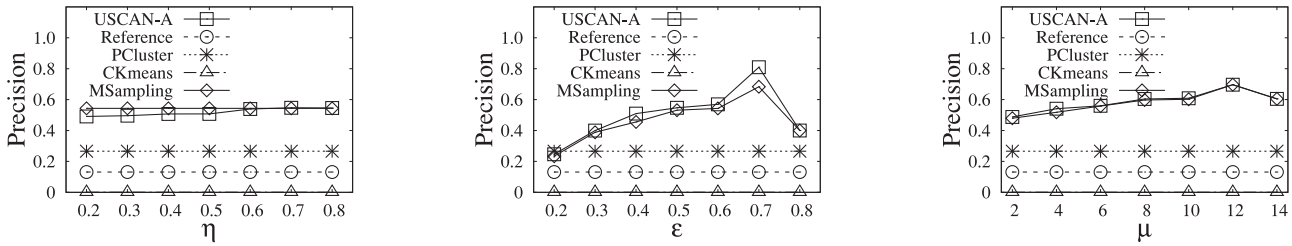
Parameter Settings. In our algorithms, we have three different parameters: η, ϵ, and μ. For both η and ϵ, we vary them from 0.2 to 0.8 with a default value of 0.5. We vary the parameter μ from 2 to 20 with a default value of 5. Unless otherwise specified, the value of the other parameter is set to its default value when varying a parameter. For the MSampling algorithm, we set the number of samples N to 150, as it performs very good when $N = 150$.

5.1 Effectiveness Testing

In this experiment, we compare the clustering qualities of our USCAN-A algorithm with those of the baseline algorithms. Following [15], we make use of the CORE dataset for testing, because we can obtain the ground truth clustering results on the CORE dataset on the basis of the MIPS protein database [15]. Based on the ground truth, we are capable of computing the number of true positives (TP), the number of false positives (FP), as well as the precision (PR = TP/(TP + FP)) obtained by a variety of algorithms. More specifically, TP is defined as the number of correctly matched interactions in predicted complexes with that in MIPS, and FP is defined as the total number of interactions in predicted complexes minus TP. We adopt the same method to compute TP, FP, and PR as used by Kollios et al. [15]. For our algorithm, we set the parameters $\eta = 0.2$, $\epsilon = 0.5$, and $\mu = 2$, since our algorithm is sufficient to achieve a good performance with this parameter setting. The results are reported in Table 2.

In Table 2, we also report the results obtained by the method proposed by Krogan et al. [21], denoted by "Reference", albeit it was shown to be much less effective than the state-of-the-art baseline [15]. As can be seen, our algorithm significantly outperforms all the baselines in terms of the clustering precision, followed by MSampling, PCluster, Reference, and CKmeans. Note that MSampling performs very good and it is even better than the state-of-the-art PCluster algorithm. However, the number of clusters identified by MSampling is much less than that of USCAN-A or PCluster. Compared to PCluster, the FP value of our algorithm USCAN-A is substantially smaller than that of PCluster, and its TP value is slightly larger than that of PCluster. The overall precision of USCAN-A is significantly higher than PCluster and MSampling. Specifically, our algorithm achieves precision 0.348, whereas the precision obtained by PCluster and MSampling is 0.266 and 0.292 respectively. Our algorithm improves the precision over PCluster and MSampling by 31 and 19 percent respectively. It is worth mentioning that the Reference algorithm is ineffective, as it yields a very large FP value. Also, we can see that the CKmeans performs extremely bad, because this method only works well for the datasets containing a small number of clusters [16]. Since CORE has many ground truth clusters, the CKmeans algorithm does not work in this dataset. These results indicate that USCAN-A is more effective than all baseline algorithms for uncertain graph clustering.

Clustering Precision with Varying Parameters. Here we study how the parameters affect the clustering qualities of our algorithm. Fig. 5 reports the precision of our algorithm with varying parameters on the CORE datasets. As can be seen, our algorithm significantly outperforms Reference, PCluster and CKmeans under most parameter settings, and it slightly outperform the MSampling algorithm. For example, in Fig. 5a, the precision of our algorithm is at least twice higher than that of PCluster with varying η. Also, we can see that the precision of our algorithm is relatively robust with respect to η. This is because the probabilities of the



(a) Vary η (b) Vary ϵ (c) Vary μ

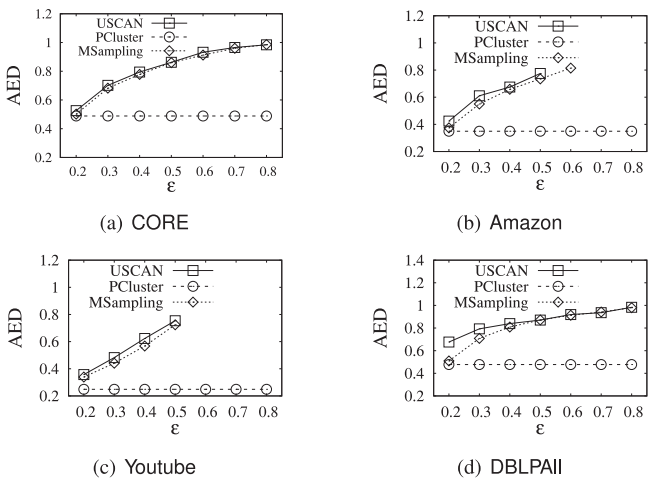
Fig. 5. Precision of our algorithm with varying parameters (CORE).

edges in the CORE datasets are very large, and thereby the resulting clusters have relatively high probabilities. As a consequence, the clusters cannot be significantly affected by the parameter η. The precision of our algorithm increases with a growing ϵ or μ. This is because with a large ϵ or μ, we may prune many unreliable edges with small probabilities, and thus the clusters will be highly reliable. When $\eta = 0.7$, $\epsilon = 0.5$, and $\mu = 12$, the precision of our algorithm is 0.70, which is much higher than that of the PCluster algorithm. These results confirm the high effectiveness of our algorithm for probabilistic graph clustering.

Average Expected Density of Different Algorithms. Here we evaluate the effectiveness of various algorithms on several large datasets. We propose a new and intuitive metric, called average expected density (AED), to measure the clustering quality for different algorithms, because all these large datasets have no ground-truth clusters. Specifically, AED is defined as

$$AED = \frac{1}{n'} \times \sum_{i=1}^{n'} \sum_{e_j \in E_i} p(e_j) \times 2/(|V_i| \times (|V_i| - 1)), \quad (8)$$

where n' denotes the number of clusters, E_i is the set of edges in the ith cluster, V_i denotes the set of nodes in the ith cluster. Intuitively, a reliable cluster should have a large AED value. A higher AED value indicates a better clustering algorithm. Similar metric is also used in [22], [23] to measure the quality of clusters in probabilistic graphs. Fig. 6 shows our results with varying ϵ. Similar results can also be obtained by varying the other parameters. From Fig. 6, we can clearly see that USCAN significantly outperforms PCluster in most parameter settings. For example, on the datasets Youtube and DBLPAll, the AED values of PCluster are very small (0.25 on Youtube and 0.5 on DBLPAll), while USCAN can achieve much higher AED values on the same datasets (0.72 on Youtube and 0.87 on DBLPAll when $\epsilon = 0.5$). It is worth mentioning that there is no ϵ parameter in the PCluster algorithm, thus its performance keeps unchanged with varying ϵ. We can also observe that USCAN is capable of finding reliable and dense clusters when ϵ increases to a reasonable value. If ϵ is too large, USCAN may return nothing, because all the vertices will be pruned given a large ϵ. In practice, $\epsilon = 0.5$ is sufficient to guarantee a good clustering results for our algorithm. Similar to the previous results, MSampling is slightly worse than USCAN, but it significantly outperforms PCluster. The results indicate that the SCAN based framework is very effective to find reliable clusters on probabilistic graphs.



(a) CORE (b) Amazon

(c) Youtube (d) DBLPAll

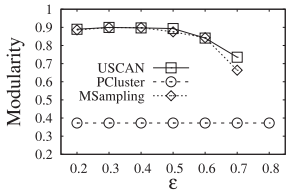
Fig. 6. Average expected density of various algorithms with varying ϵ.

Expected Modularity of Various Algorithms. Here we use the modularity-based metric to measure the effectiveness of our algorithm. The concept of modularity was proposed by Newman [24], [25] which can be used to measure the clustering quality. Based on the well-known modularity measure, we define a measure called expected modularity as follows:

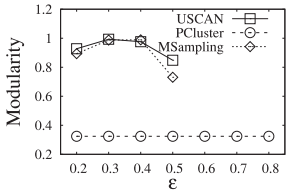
$$\bar{Q} = \frac{1}{\mathcal{N}} \times \sum_{G \in \mathcal{G}} Q_G, \quad (9)$$

where G is a possible world of the probabilistic graph $\mathcal{G}$, $\mathcal{N}$ denotes the number of all possible worlds of $\mathcal{G}$, Q_G denotes the modularity of the possible world G [25]. Note that the modularity of a possible world G can be easily calculated by definition [25]. Since $\mathcal{N}$ is exponentially large, we use a sampling based algorithm to estimate the expected modularity as follows. Specifically, we first sample N possible worlds and then compute the standard modularity on each possible world. Subsequently, we take the average value of modularity over all N possible worlds. Fig. 7 shows the expected modularity of USCAN, PCluster and MSampling on CORE and Amazon datasets. Similar results can also be observed on the other datasets. As can be seen, USCAN significantly outperforms PCluster and it is also slightly better than MSampling. These results further confirm the effectiveness of our algorithm.

Sensitive Analysis. In this experiment, we study whether our USCAN algorithm is sensitive to the edge insertion or not. To this end, we first randomly insert 100 edges into the probabilistic graph with a fixed probability p selected from a range $[0.1, 0.9]$. Then, we run USCAN on this slightly modified probabilistic graph and evaluate the clustering quality of USCAN based on the average expected density and expected modularity. Fig. 8 shows the results on CORE datasets. Similar results can also be observed on the other datasets. Note that in Fig. 8, USCAN + denotes the result of

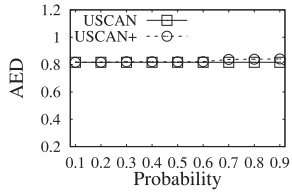


(a) CORE

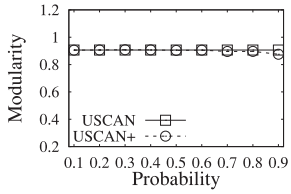


(b) Amazon

Fig. 7. Expected modularity of various algorithms with varying ϵ.



(a) AED



(b) Modularity

Fig. 8. Sensitive analysis of USCAN on CORE.

USCAN on the modified probabilistic graph. As can be seen, USCAN is very robust with respect to the probability p. Even when $p = 0.9$, USCAN can still obtain similar results on both the original and modified probabilistic graph. These results indicate that our algorithm is robust under a small perturbation of the probabilistic graph.

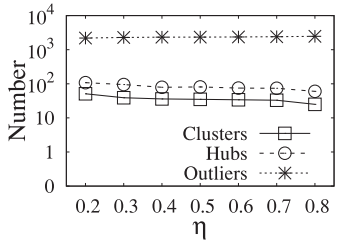
Statistics of the Reliable Structural Clustering. Here we report the statistics of the reliable structural clustering results. Fig. 9 show the number clusters, hubs, and outliers on CORE and DBLPAll with varying parameters separately. Similar results can also be observed on the other datasets. In general, both the number of clusters and the number of hubs decrease with increasing η, ϵ, and μ. The number of outliers increases when η, ϵ, and μ increase. This is because by the definitions of clusters and hubs in Section 2, our algorithm is able to prune many vertices for large η, ϵ and μ values. As a result, the number of clusters and hubs decreases and the number of outliers increases, with increasing η, ϵ and μ.

5.2 Runtime Testing

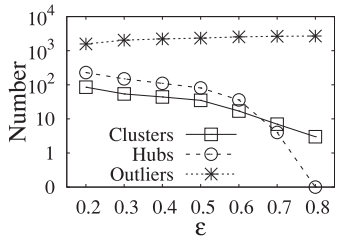
In this section, we first evaluate the runtime of different algorithms. Then, we perform comprehensive experiments to test the runtime of our algorithms with varying parameters.

Runtime of Various Algorithms. We compare the running time of our best algorithm (USCAN-A) with those of the baselines. The parameters of USCAN-A are set as their default values ($\eta = 0.5$, $\epsilon = 0.5$, and $\mu = 5$). Fig. 10 reports the time consumption of various algorithms. As can be seen, USCAN-A, MSampling and PCluster are very efficient on all datasets, whereas the CKmeans algorithm is very costly for handling large graphs. On the CORE, DBLP01 and Amazon datasets, our algorithm is significantly faster than MSampling, PCluster and CKmeans. In the DBLPAll and Youtube datasets, PCluster is faster than USCAN-A. It should be noted that although PCluster is more efficient than our algorithm on large datasets, its clustering quality is significantly worse than our algorithm, as demonstrated in the previous experiments. Our algorithm not only produces high-quality clustering results, but it also very efficient to handle large real-world probabilistic graphs. Note that MSampling is very costly, because it needs to evaluate the structural similarities for all edges in each possible world. In addition, it is worth mentioning that CKmeans cannot complete in one day on the DBLPAll datasets due to the high time and space complexities of the algorithm.

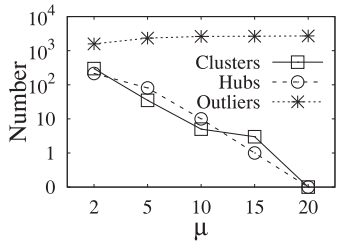
Runtime Results: Varying η. Fig. 11 reports the runtime of USCAN-B, USCAN-E, and USCAN-A by varying η on four different datasets. Similar results can also be observed on DBLPAll. From Fig. 11, we can clearly see that USCAN-A outperforms the other two algorithms, since it is integrated all the pruning rules. On large datasets, USCAN-A can even be one order of magnitude faster than USCAN-E. For instance, on the Youtube dataset, USCAN-A is more than 20 times faster than USCAN-E. Also, we can see that USCAN-E is significantly faster than USCAN-B, due to the powerful early-termination pruning (see Section 4). The USCAN-B algorithm is very costly for handling large datasets. For example, USCAN-B cannot be completed in one day in the Youtube dataset. Generally, the time costs of our algorithms decrease with an increasing η. The reason is that the larger η is, the more edges will be pruned by our algorithms. These results confirm our theoretical analysis in the previous sections.



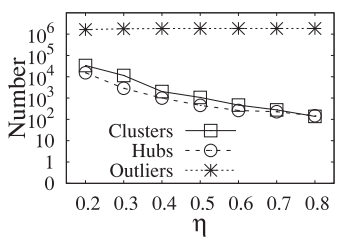
(a) Vary η (CORE)



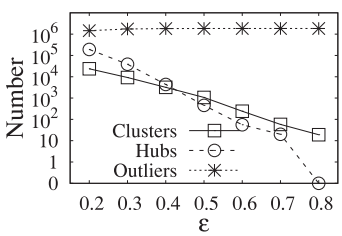
(b) Vary ϵ (CORE)



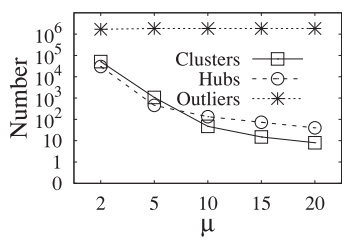
(c) Vary μ (CORE)



(d) Vary η (DBLPAll)



(e) Vary ϵ (DBLPAll)



(f) Vary μ (DBLPAll)

Fig. 9. The number of clusters, hubs, and outliers with varying parameters.

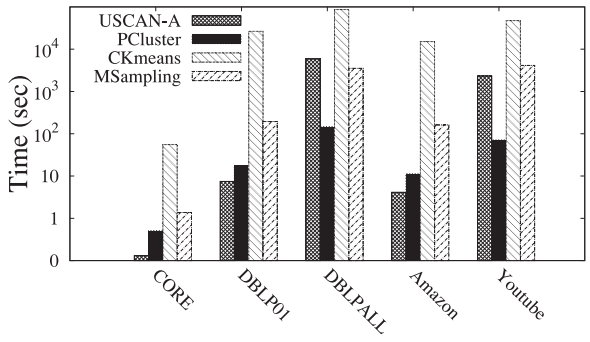


Fig. 10. Runtime of different algorithms.

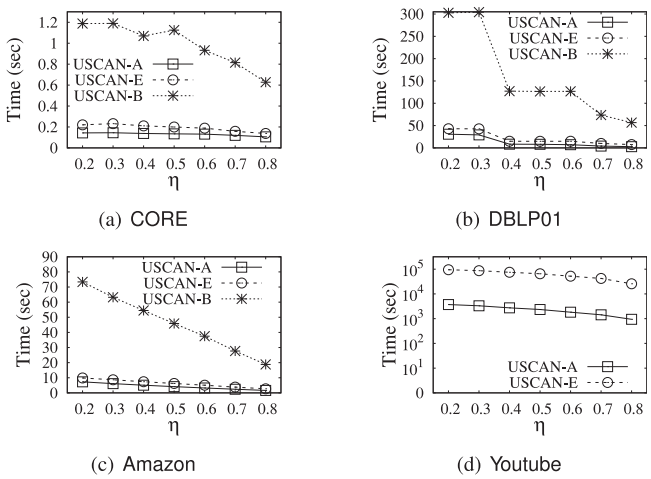


Fig. 11. Runtime of our algorithms with varying η.

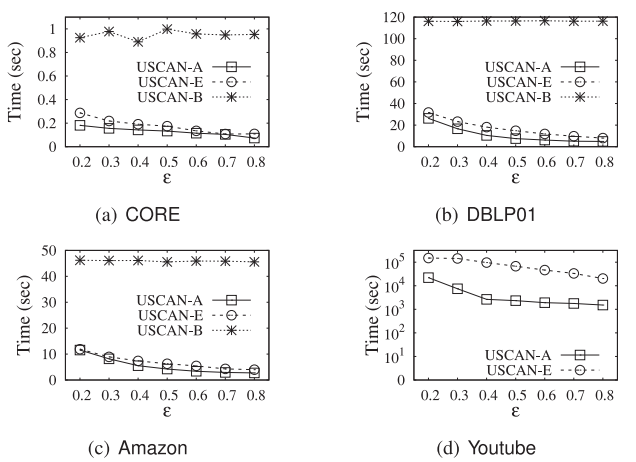


Fig. 12. Runtime of our algorithms with varying ϵ.

Runtime Results: Varying ϵ. We evaluate the efficiency of our algorithms with varying ϵ on four diverse datasets. The results of varying ϵ are shown in Fig. 12. Again, similar results can be obtained on DBLPAll. As can be seen, USCAN-A is clearly the winner among all our algorithms, followed by USCAN-E and USCAN-B. The time costs of both USCAN-A and USCAN-E decrease with increasing ϵ, while the time overhead of USCAN-B is relatively robust w.r.t. ϵ. This is because the early-termination pruning of both USCAN-A and USCAN-E performs well for a large ϵ. The USCAN-B algorithm, however, does not use this pruning rule, thus it is robust w.r.t. ϵ. Also, we can see that USCAN-A is at least one order of magnitude faster than USCAN-E on large datasets. These results further demonstrate the high efficiency of USCAN-A.

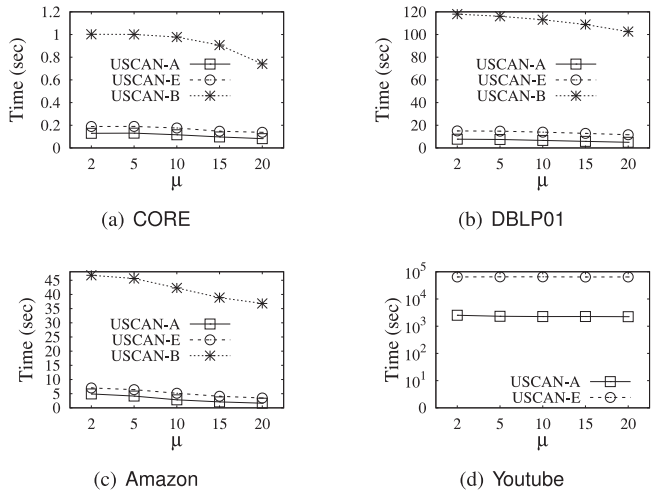


Fig. 13. Runtime of our algorithms with varying μ.

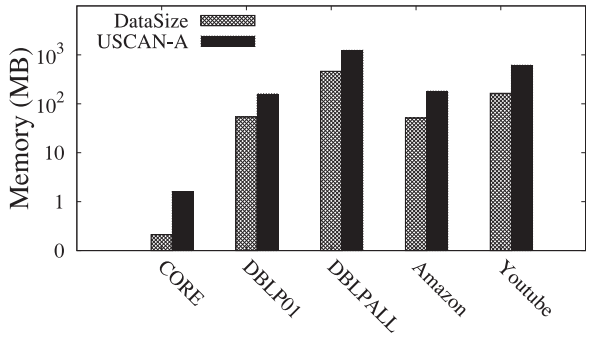


Fig. 14. Memory overhead of USCAN-A.

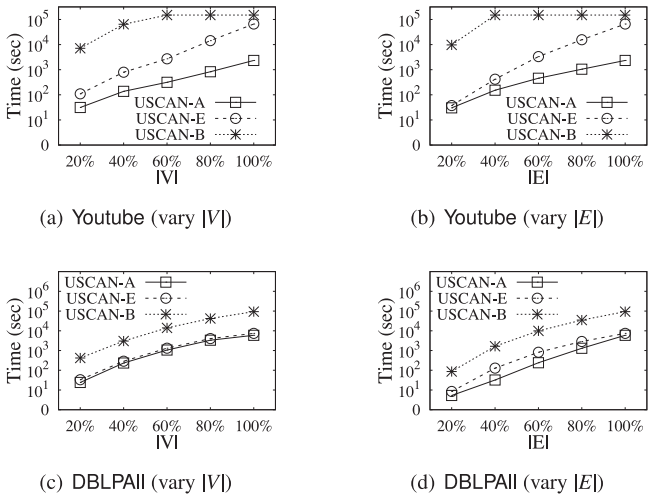
Runtime Results: Varying μ. Here we show how the parameter μ affects the efficiency of our algorithms. Fig. 13 shows our results on four different datasets, and similar results can also be observed on DBLPAll. Similarly, USCAN-A is better than USCAN-E and USCAN-B. In general, the time spent by our algorithms decreases as μ increases. This is because, our algorithms can prune many non-core vertices with a large μ, thus significantly reducing the computational costs of our algorithms. These results further validate our theoretical results presented in the previous sections.

5.3 Memory Consumption and Scalability Testing

In this section, we evaluate the memory usage and the scalability of our algorithms.

Memory Consumption. We report the memory overhead of USCAN-A in Fig. 14. As shown in Fig. 14, the space cost of our algorithm is around three times of the size of the graph on all five datasets. For example, on DBLPAll, USCAN-A uses 160 MB memory, while the graph size is 54 MB. This is because the space complexity of our algorithm is nearly linear w.r.t. the graph size. These results suggest that USCAN-A is highly space-efficient, confirming the theoretical analysis in Section 3.

Scalability Testing. Here we evaluate the scalability of our algorithms using two large datasets: Youtube and DBLPAll. For each dataset, we generate four subgraphs by randomly sample 20-80 percent of the vertices and 20-80 percent of the edges, respectively. In this experiment, we set the parameters of our algorithm to be their default values. Similar results can



(a) Youtube (vary $|V|$) (b) Youtube (vary $|E|$)

(c) DBLPAll (vary $|V|$) (d) DBLPAll (vary $|E|$)

Fig. 15. Scalability testing.

also be observed with other parameter settings. The results are shown in Fig. 15. As can be observed, the running time of both USCAN-A and USCAN-E increase smoothly w.r.t. $|V|$ or $|E|$, indicating that our algorithms scale well in real-world graphs. The USCAN-B algorithm, however, cannot handle large datasets. These results indicate that the pruning techniques developed in USCAN-A and USCAN-E are very powerful, resulting in the high scalability of our algorithms.

6 Related Work

Besides the probabilistic graph clustering, our work is also related to cohesive subgraph mining in uncertain graphs and the deterministic graph clustering. Below, we review the existing studies on these topics.

Cohesive Subgraphs in Probabilistic Graphs. In the literature, there are many cohesive subgraph models are proposed for deterministic graphs, such as maximal clique [26], [27], [28], [29], quasi-clique [30], k-core [31], [32], and k-truss [33], [34]. Recently, similar cohesive subgraph concepts are extended to probabilistic graphs. Bonchi et al. extended the concept of k-core to probabilistic semantics [35]. Mehmood et al. focused on the influence cascade problem under a certain contagion model, in which the influence is taken as a reachability metric to measure whether or not a node can be activated or influenced [36]. Unlike their work, we concentrates mainly on the reliable structural clustering problem. Zou et al. [37] studied a top-k maximal clique search problem in uncertain graphs. More recently, Mukherjee et al. [18] proposed a new uncertain maximal clique model as well as an efficient uncertain maximal clique enumeration algorithm. Gao et al. [38] proposed a solution on finding RkNN over uncertain graphs. Huang et al. [39] proposed the definitions of local and global (k,γ)-truss which enable the truss decomposition on uncertain graphs. They also proposed a dynamic programming algorithm to compute the local k-truss. However, their DP algorithm cannot be directly used for solving our problem, because the reliable structural similarity computation in our problem is much more complicated than computing the local k-truss.

Deterministic Graph Clustering. In the literature, there are a huge number of clustering methods on deterministic graphs, such as modularity-based method [40], [41], [42], graph partitioning [43], [44], [45], and density-based method [46]. An excellent survey on deterministic graph clustering has been made by Aggarwal et al. [47]. Also, some graph summarization techniques can also be used for clustering. A good review on this topic can be found in [48], [49]. Xu et al. [1] proposed a clustering method called Structural Clustering Algorithm for Networks (SCAN). Different from the previous works, SCAN can find out the clusters as well as the hubs and the outliers. Hubs and outliers are vertices which do not belong to any clusters. A hub is a vertex that bridges two or more clusters, and an outlier is a vertex which connects to one or zero cluster. To improve the efficiency of SCAN, [3] and [4] proposed several pruning rules to speed up the clustering process significantly. All these SCAN algorithms cannot directly used for probabilistic graph data. In this paper, we extend the SCAN framework to probabilistic graph on the basis of a newly-proposed concept called reliable structural similarity.

7 Conclusion

In this paper, we study the structural clustering problem on probabilistic graphs. Unlike the existing structural clustering problems on deterministic graphs, our problem relies on a new concept called reliable structural similarity, which is used to measure the probability of similarity between two vertices in the probabilistic graph. We develop a new dynamic programming algorithm with several carefully-designed pruning techniques to efficiently compute the reliable structural similarities. With the reliable structural similarities, we extend the state-of-the-art structural clustering algorithm to efficiently solve our problem. Extensive experiments on five real-life datasets show the effectiveness, efficiency, and scalability of the proposed solutions.

Acknowledgments

This work was partially supported by (i) the National Key R&D Program of China 2018YFB1003201; (ii) NSFC Grants 61772346, 61732003, 61772091, and 61802035; (iii) the Beijing Institute of Technology Research Fund Program for Young Scholars; (iv) the Research Grants Council of the Hong Kong SAR, China No. 14221716; (v) ARC Discovery Project Grant DP160102114; and (vi) Guangdong Pre-national project 2014GKXM054.

References

[1] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "SCAN: A structural clustering algorithm for networks," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 824–833.

[2] S. Lim, S. Ryu, S. Kwon, K. Jung, and J. Lee, "LinkSCAN*: Overlapping community detection using the link-space transformation," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 292–303.

[3] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "SCAN++: Efficient algorithm for finding clusters, hubs and outliers on large-scale graphs," *Proc. VLDB Endowment*, vol. 8, no. 11, pp. 1178–1189, 2015.

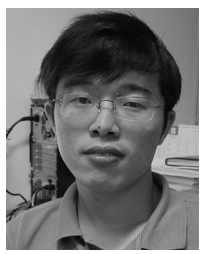
[4] L. Chang, W. Li, X. Lin, L. Qin, and W. Zhang, "pSCAN: Fast and exact structural graph clustering," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, May 2016, pp. 253–264.

[5] C. C. Aggarwal, *Managing and Mining Uncertain Data*, vol. 35. Norwell, MA, USA: Kluwer, 2009.

[6] J. S. Bader, A. Chaudhuri, J. M. Rothberg, and J. Chant, "Gaining confidence in high-throughput protein interaction networks," *Nature Biotechnology*, vol. 22, no. 1, pp. 78–85, 2004.

[7] H. Kawahigashi, Y. Terashima, N. Miyauchi, and T. Nakakawaji, "Modeling ad hoc sensor networks using random graph theory," in *Proc. 2nd IEEE Consum. Commun. Netw. Conf.*, 2005, pp. 104–109.
[8] U. Kuter and J. Golbeck, "SUNNY: A new algorithm for trust inference in social networks using probabilistic confidence models," in *Proc. 22nd Nat. Conf. Artif. Intell.*, 2007, pp. 1377–1382.
[9] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 137–146.
[10] P. Hintsanen, "The most reliable subgraph problem," in *Proc. Eur. Conf. Principles Data Mining Knowl. Discovery*, 2007, pp. 471–478.
[11] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "k-nearest neighbors in uncertain graphs," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 997–1008, 2010.
[12] R. Jin, L. Liu, B. Ding, and H. Wang, "Distance-constraint reachability computation in uncertain graphs," *Proc. VLDB Endowment*, vol. 4, no. 9, pp. 551–562, 2011.
[13] Z. Zou, H. Gao, and J. Li, "Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 633–642.
[14] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding top-k maximal cliques in an uncertain graph," in *Proc. IEEE 26th Int. Conf. Data Eng.*, 2010, pp. 649–652.
[15] G. Kollios, M. Potamias, and E. Terzi, "Clustering large probabilistic graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 2, pp. 325–336, Feb. 2013.
[16] L. Liu, R. Jin, C. Aggarwal, and Y. Shen, "Reliable clustering on uncertain graphs," in *Proc. IEEE 12th Int. Conf. Data Mining*, 2012, pp. 459–468.
[17] R. Jin, L. Liu, and C. C. Aggarwal, "Discovering highly reliable subgraphs in uncertain graphs," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 992–1000.
[18] A. P. Mukherjee, P. Xu, and S. Tirthapura, "Mining maximal cliques from an uncertain graph," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 243–254.
[19] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, 1985.
[20] M. C. Lin, F. J. Soulignac, and J. L. Szwarcfiter, "Arboricity, h-index, and dynamic algorithms," *Theoretical Comput. Sci.*, vol. 426, pp. 75–90, 2012.
[21] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, et al., "Global landscape of protein complexes in the yeast saccharomyces cerevisiae," *Nature*, vol. 440, no. 7084, pp. 637–643, 2006.
[22] S. E. Schaeffer, "Graph clustering," *Comput. Sci. Rev.*, vol. 1, no. 1, pp. 27–64, 2007.
[23] B. Zhao, J. Wang, M. Li, F.-X. Wu, and Y. Pan, "Detecting protein complexes based on uncertain graph model," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 11, no. 3, pp. 486–497, May/Jun. 2014.
[24] M. E. J. Newman, "The structure and function of complex networks," *SIAM Rev.*, vol. 45, no. 2, pp. 167–256, 2003.
[25] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Nat. Acad. Sci. United States America*, vol. 103, no. 23, pp. 8577–8582, 2006.
[26] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973.
[27] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by H*-graph," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 447–458.
[28] J. Wang, J. Cheng, and A. W.-C. Fu, "Redundancy-aware maximal cliques," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 122–130.
[29] J. Xiang, C. Guo, and A. Aboulnaga, "Scalable maximum clique computation using MapReduce," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 74–85.
[30] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, "Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 104–112.
[31] V. Batagelj and M. Zaversnik, "An O (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.
[32] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 51–62.
[33] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," Nat. Secur. Agency Tech. Rep., 2008, Art. no. 16.
[34] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 812–823, 2012.
[35] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich, "Core decomposition of uncertain graphs," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1316–1325.
[36] Y. Mehmood, F. Bonchi, and D. García-Soriano, "Spheres of influence for more effective viral marketing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 711–726.
[37] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding top-k maximal cliques in an uncertain graph," in *Proc. IEEE 26th Int. Conf. Data Eng.*, 2010, pp. 649–652.
[38] Y. Gao, X. Miao, G. Chen, B. Zheng, D. Cai, and H. Cui, "On efficiently finding reverse k-nearest neighbors over uncertain graphs," *The VLDB J.*, vol. 26, pp. 1–26, 2017.
[39] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 77–90.
[40] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, 2004, Art. no. 066111.
[41] R. Guimera and L. A. N. Amaral, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, no. 7028, pp. 895–900, 2005.
[42] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, 2004, Art. no. 026113.
[43] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 107–114.
[44] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.
[45] L. Wang, Y. Xiao, B. Shao, and H. Wang, "How to partition a billion-node graph," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 568–579.
[46] P. Jiang and M. Singh, "SPICi: A fast clustering algorithm for large biological networks," *Bioinf.*, vol. 26, no. 8, pp. 1105–1111, 2010.
[47] C. C. Aggarwal and H. Wang, "A survey of clustering algorithms for graph data," in *Managing and Mining Graph Data*. Berlin, Germany: Springer, 2010, pp. 275–301.
[48] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, 2018, Art. no. 62.
[49] N. Hassanlou, M. Shoaran, and A. Thomo, "Probabilistic graph summarization," in *Proc. 14th Int. Conf. Web-Age Inf. Manage.*, 2013, pp. 545–556.

Yu-Xuan Qiu received the BE and ME degrees in computer science from Shenzhen University, in 2015 and 2018, respectively. He is currently a research assistant at the Beijing Institute of Technology, Beijing, China. His current research interests include graph data management and mining.

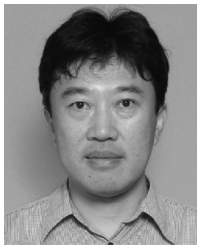


Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong, in 2013. He is currently an associate professor at the Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

Jianxin Li received the PhD degree in computer science from the Swinburne University of Technology, Australia, in 2009. He is a senior lecturer with the School of Computer Science and Software Engineering, University of Western Australia. His research interests include database query processing and optimization, social network analysis, and traffic network data processing.

Shaojie Qiao received the BS and PhD degrees from Sichuan University, Chengdu, China, in 2004 and 2009, respectively. From 2007 to 2008, he was a visiting scholar with the School of Computing, National University of Singapore. He is currently a professor with the School of Cybersecurity, Chengdu University of Information Technology, Chengdu. He has led several research projects in moving objects databases and trajectory data mining. He authored more than 40 high-quality papers and co-authored more than 90 papers. His research interests include trajectory prediction and intelligent transportation systems.

Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991, and 1996, respectively. Currently, he is a professor with the Department of Computer Science, Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.



Jeffery Xu Yu received the BE, ME, and PhD degrees in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He has held teaching positions at the Institute of Information Sciences and Electronics, University of Tsukuba, and with the Department of Computer Science, Australian National University, Australia. Currently, he is a professor with the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong, Hong Kong. His current research interests include graph database, graph mining, keyword search in relational databases, and social network analysis.

Rui Mao received the PhD degree in computer science from the University of Texas at Austin, in 2007. He is currently a professor in Shenzhen University. His research interests include big data analysis and management, content-based similarity query of multimedia and biological data, data mining, and machine learning.