Data Min Knowl Disc (2012) 25:450–477
DOI 10.1007/s10618-012-0263-0

Clustering large attributed information networks: an efficient incremental computing approach

Hong Cheng · Yang Zhou · Xin Huang · Jeffrey Xu Yu

Received: 30 April 2011 / Accepted: 16 February 2012 / Published online: 4 March 2012

Abstract In recent years, many information networks have become available for analysis, including social networks, road networks, sensor networks, biological networks, etc. Graph clustering has shown its effectiveness in analyzing and visualizing large networks. The goal of graph clustering is to partition vertices in a large graph into clusters based on various criteria such as vertex connectivity or neighborhood similarity. Many existing graph clustering methods mainly focus on the topological structures, but largely ignore the vertex properties which are often heterogeneous. Recently, a new graph clustering algorithm, *SA-Cluster*, has been proposed which combines structural and attribute similarities through a unified distance measure. SA-Cluster performs matrix multiplication to calculate the random walk distances between graph vertices. As part of the clustering refinement, the graph edge weights are iteratively adjusted to balance the relative importance between structural and attribute similarities. As a consequence, matrix multiplication is repeated in each iteration of the clustering process to recalculate the random walk distances which are affected by the edge weight

Responsible editor: Fei Wang, Hanghang Tong, Phillip Yu, Charu Aggarwal.

H. Cheng (✉) · X. Huang · J. X. Yu
Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, China
e-mail: hcheng@se.cuhk.edu.hk

X. Huang
e-mail: xhuang@se.cuhk.edu.hk

J. X. Yu
e-mail: yu@se.cuhk.edu.hk

Y. Zhou
College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
e-mail: yzhou@gatech.edu

update. In order to improve the efficiency and scalability of SA-Cluster, in this paper, we propose an efficient algorithm *Inc-Cluster* to incrementally update the random walk distances given the edge weight increments. Complexity analysis is provided to estimate how much runtime cost Inc-Cluster can save. We further design parallel matrix computation techniques on a multicore architecture. Experimental results demonstrate that Inc-Cluster achieves significant speedup over SA-Cluster on large graphs, while achieving exactly the same clustering quality in terms of intra-cluster structural cohesiveness and attribute value homogeneity.

Keywords Graph clustering · Incremental computation · Parallel computing

1 Introduction

Graphs are popularly used to model structural relationship between objects in many application domains such as the Web, social networks, sensor networks, biological networks and communication networks, etc. Graph clustering has received a lot of attention recently with many proposed clustering algorithms (Shi and Malik 2000, Newman and Girvan 2004, Xu et al. 2007, Satuluri and Parthasarathy 2009, Zhou et al. 2009). Clustering on a large graph aims to partition the graph into several densely connected components. Typical applications of graph clustering include community detection in social networks, identification of functional modules in large protein-protein interaction networks, etc. Many existing graph clustering methods mainly focus on the topological structure of a graph so that each partition achieves a cohesive internal structure. Such methods include clustering based on normalized cuts (Shi and Malik 2000), modularity (Newman and Girvan 2004), structural density (Xu et al. 2007) or flows (Satuluri and Parthasarathy 2009). On the other hand, a recent graph summarization method (Tian et al. 2008) aims to partition a graph according to attribute similarity, so that nodes with the same attribute values are grouped into one partition.

In many real applications, both the graph topological structure and the vertex properties are important. For example, in a social network, vertex properties describe roles of a person while the topological structure represents relationships among a group of people. The graph clustering and summarization approaches mentioned above consider only one aspect of the graph properties but ignore the other. As a result, the clusters thus generated would either have a rather random distribution of vertex properties within clusters, or have a rather loose intra-cluster structure. An ideal graph clustering should generate clusters which have a *cohesive* intra-cluster structure with *homogeneous* vertex properties, by balancing the structural and attribute similarities.

Figure 1 shows an example of a coauthor graph where a vertex represents an author and an edge represents the coauthor relationship between two authors. In addition, there are an author ID, research topic and age associated with each author. The research topic and age are considered as attributes to describe the vertex properties. As we can see, authors r_1–r_7 work on XML, authors r_9–r_{11} work on skyline and r_8 works on both. In addition, each author has a range value to describe his/her age. The problem studied in this paper is to cluster a graph associated with attributes (called *an attributed graph*), such as the example in Fig. 1, based on both structural and attribute similarities. The

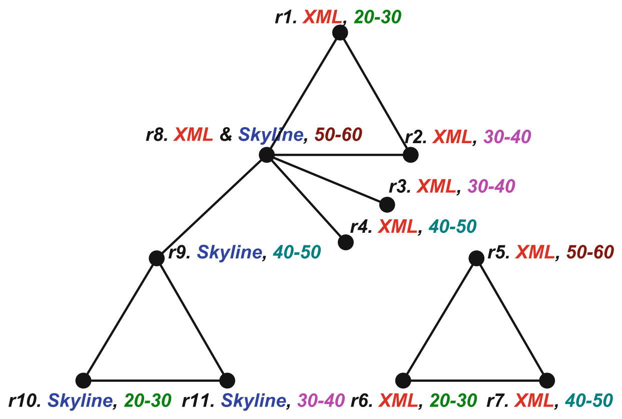


Fig. 1 A Coauthor network with two attributes "Topic" and "Age"

goal is to partition the graph into k clusters with cohesive intra-cluster structures and homogeneous attribute values. The problem is quite challenging because structural and attribute similarities are two seemingly independent, or even conflicting goals—in our example, authors who collaborate with each other may have different values on research topics and age; while authors who work on the same topics or who are in a similar age may come from different groups with no collaborations. It is not straightforward to balance these two objectives.

In a recent work, Zhou et al. have proposed SA-Cluster (2009), a graph clustering algorithm by combining structural and attribute similarities. A set of attribute vertices and attribute edges are added to the original graph. With such graph augmentation, the attribute similarity is transformed to vertex proximity in the graph—two vertices which share an attribute value are connected by a common attribute vertex. A neighborhood random walk model, which measures the vertex closeness on the augmented graph through both structure edges and attribute edges, unifies the two similarities. Then SA-Cluster uses the random walk distance as the vertex similarity measure and performs clustering by following the K-Medoids framework. As different attributes may have different degrees of importance, a weight ω_i, which is initialized to 1.0, is assigned to the attribute edges corresponding to attribute a_i. The attribute edge weights $\{\omega_1, \ldots, \omega_m\}$ are updated in each iteration of the clustering process, to reflect the importance of different attributes. In the above example, after the first iteration, the weight of research topic will be increased to a larger value while the weight of age will be decreased, as research topic has better clustering tendency than age. Accordingly, the transition probabilities on the graph are affected with the attribute weight adjustments. Thus the random walk distance matrix needs to be *recalculated in each iteration of the clustering process*. Since the random walk distance calculation involves matrix multiplication, which has a time complexity of $O(n^3)$, the repeated random walk distance calculation causes a non-trivial computational cost in SA-Cluster. We find in the experiments that the random walk distance computation takes 98% of the total clustering time in SA-Cluster.

With a careful study of the weight self-adjustment mechanism in Zhou et al. (2009), we observed that the weight increments only affect the attribute edges in the augmented graph, while the structure edges are not affected. Motivated by this, in this paper, we aim to improve the efficiency and scalability of SA-Cluster with a proposed efficient incremental computation algorithm Inc-Cluster to update the random walk distance matrix. A preliminary study on this problem appeared in Zhou et al. (2010). The core idea is to compute the full random walk distance matrix only once at the beginning of the clustering process. Then in each following iteration of clustering, given the attribute weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$, we use Inc-Cluster to update the original random walk distance matrix, instead of re-calculating the matrix from scratch. This incremental computation problem is quite challenging. Existing incremental approaches to compute PageRank-style scores (Desikan et al. 2005; Wu and Raschid 2009) cannot be directly applied to solve our problem, as they partition the graph into a changed part and an unchanged part apriori. But in our problem it is hard to find such a clear boundary between the changed and the unchanged parts on the graph, because the effect of edge weight adjustments is propagated widely to the whole graph in multiple steps. The distance between any pair of vertices may be affected. In our solution, we examine the structure edges and the attribute edges separately to see how they would be affected by the attribute weight adjustments. Accordingly, we partition the random walk matrix into four submatrices and incrementally update the matrix elements which are affected by the weight adjustments. With the proposed Inc-Cluster algorithm, we can divide the graph clustering algorithm into two phases: an offline phase at the beginning of clustering for the full random walk distance matrix computation which is relatively expensive, and an online phase for the fast iterative clustering process with the incremental matrix calculation which is much cheaper. The main contributions of this paper are summarized below.

1. We study the problem of incremental computation of the random walk distance matrix in the context of graph clustering with structural and attribute similarities. We propose an efficient algorithm Inc-Cluster to incrementally update the random walk distance matrix given the attribute weight increments. By analyzing how the transition probabilities are affected by the weight increments, the random walk distance matrix is divided into submatrices for incremental update. We also design parallel matrix computation techniques on a multicore architecture for further speed improvement. Importantly, the incremental approach is also applicable to fast random walk computation in continuously evolving graphs with vertex/edge insertions and deletions.
2. Complexity analysis is provided to quantitatively estimate the upper bound and the lower bound of the number of elements in the random walk distance matrix that remain unchanged. The upper bound and lower bound correspond to the best case and the worst case of the incremental approach respectively. An analysis on the storage cost of the incremental algorithm is also provided, which shows that Inc-Cluster requires a small amount of extra space compared with SA-Cluster.
3. We perform extensive evaluation of the incremental approach on real large graphs, demonstrating that our method Inc-Cluster is able to achieve significant speedup over SA-Cluster, and the parallel version of Inc-Cluster can further reduce the

runtime by 52–62% over the non-parallel version. At the same time, Inc-Cluster achieves exactly the same clustering quality in terms of intra-cluster structural cohesiveness and attribute value homogeneity.

The rest of the paper is organized as follows. We review related work on graph clustering and graph mining in Sect. 2. Section 3 introduces preliminary concepts and analyzes the runtime cost of SA-Cluster. Section 4 presents our proposed incremental algorithm Inc-Cluster. Time complexity analysis is provided in Sect. 5, followed by a discussion on the storage cost of Inc-Cluster in Sect. 6. Section 7 presents extensive experimental results. Finally, Sect. 8 concludes the paper.

2 Related work

Many graph clustering techniques have been proposed which mainly focused on the topological structures based on various criteria including normalized cuts (Shi and Malik 2000), modularity (Newman and Girvan 2004), structural density (Xu et al. 2007) or stochastic flows (Satuluri and Parthasarathy 2009). The clustering results contain densely connected components within clusters. However, such methods usually ignore vertex attributes in the clustering process. On the other hand, Tian et al. (2008) proposed OLAP-style aggregation approaches to summarize large graphs by grouping nodes based on user-selected attributes. This method achieves homogeneous attribute values within clusters, but ignores the intra-cluster topological structures. Recently, Zhou et al. have proposed a graph clustering algorithm, SA-Cluster (2009), based on both structural and attribute similarities. Experimental results have shown that SA-Cluster achieves a good balance between structural cohesiveness and attribute homogeneity. Long et al. (2006) proposed a collective factorization model for multi-type relational data clustering. In their model, multiple types of objects and their interrelations are considered, in addition to the object features. A spectral relational clustering is proposed to cluster multi-type interrelated data objects simultaneously. The main differences between Inc-Cluster (and SA-Cluster) and the spectral clustering algorithm (Long et al. 2006) include: (1) Inc-Cluster (and SA-Cluster) considers one type of objects and their connections, while (Long et al. 2006) considers the interrelations between different types of objects; and (2) when combining the connections and feature similarities, Long et al. (2006) has to provide a set of weights $\omega_a^{(ij)}$ and $\omega_b^{(i)}$ for different types of objects, while Inc-Cluster (and SA-Cluster) designs a weight self-adjusting mechanism to automatically adjust the attribute weights.

Other recent studies on graph clustering include the following. Sun et al. (2007) proposed GraphScope which is able to discover communities in large and dynamic graphs, as well as to detect the changing time of communities. Wang et al. (2011) used nonnegative matrix factorization to find the communities in networks because of its powerful interpretability and close relationship between clustering methods. Sun et al. (2009) proposed an algorithm, RankClus, which integrates clustering with ranking in large-scale information network analysis. The final results contain a set of clusters with a ranking of objects within each cluster. Navlakha et al. (2008) proposed a graph summarization method using the MDL principle. Cai et al. (2005) proposed an algorithm for mining communities on heterogeneous social networks. Tsai and

Chiu (2008) developed a feature weight self-adjustment mechanism for K-Means clustering on relational datasets. In that study, finding feature weights is modeled as an optimization problem to minimize the separations within clusters and maximize the separations between clusters. The adjustment margin of a feature weight is estimated by the importance of the feature in clustering.

The concept of random walk has been widely used to measure vertex distances. Jeh and Widom (2002) designed a measure called SimRank, which defines the similarity between two vertices in a graph by their neighborhood similarity. Pons and Latapy (2006) proposed to use short random walks of length l to measure the similarity between two vertices in a graph for community detection. Tong et al. (2006; 2008) proposed an algorithm for fast random walk computation, by partitioning a graph into k clusters apriori and then decomposing the transition probability matrix into a within-partition one and a cross-partition one. There are also some studies for incremental computation of PageRank scores. Desikan et al. (2005) proposed an incremental algorithm to compute PageRank for the evolving Web graph by partitioning the graph into a changed part and an unchanged part. Wu and Raschid (2009) computes the local PageRank scores on a subgraph by assuming that the scores of external pages are known.

3 Preliminary concepts

In this section, we first introduce the problem formulation of graph clustering considering both structural and attribute similarities. We then give a brief review of an earlier algorithm SA-Cluster by Zhou et al. (2009) and analyze the computational cost. Our proposed approach to handle the computational bottleneck is outlined.

3.1 Attribute augmented graph

Definition 1 (*Attributed graph*) An attributed graph is denoted as $G = (V, E, \Lambda)$, where V is the set of vertices, E is the set of edges, and $\Lambda = \{a_1, \ldots, a_m\}$ is the set of attributes associated with vertices in V for describing vertex properties. A vertex $v \in V$ is associated with an attribute vector $[a_1(v), \ldots, a_m(v)]$ where $a_j(v)$ is the attribute value of vertex v on attribute a_j.

Attributed graph clustering is to partition an attributed graph G into k disjoint subgraphs $\{G_i = (V_i, E_i, \Lambda)\}_{i=1}^{k}$, where $V = \bigcup_{i=1}^{k} V_i$ and $V_i \bigcap V_j = \emptyset$ for any $i \neq j$. A desired clustering of an attributed graph should achieve a good balance between the following two objectives: (1) vertices within one cluster are close to each other in terms of structure, while vertices between clusters are distant from each other; and (2) vertices within one cluster have similar attribute values, while vertices between clusters could have quite different attribute values.

Zhou et al. (2009) proposed an *attribute augmented graph* to represent attributes explicitly as *attribute vertices and edges*. In this paper we follow the same representation.

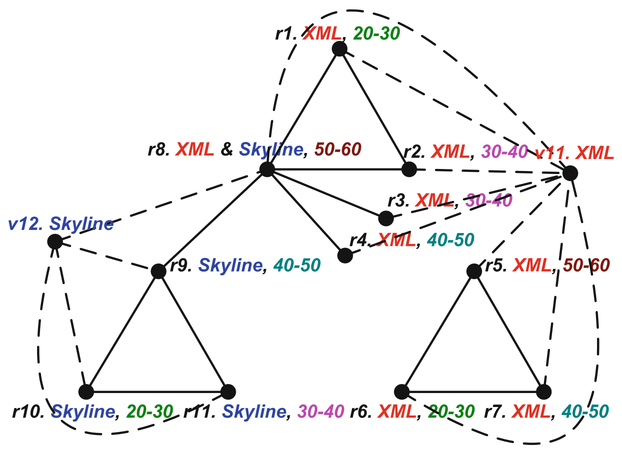


Fig. 2 Attribute augmented graph

Definition 2 (*Attribute augmented graph*) Given an attributed graph $G = (V, E, \Lambda)$ with a set of attributes $\Lambda = \{a_1, \ldots, a_m\}$. The domain of attribute a_i is $Dom(a_i) = \{a_{i1}, \ldots, a_{in_i}\}$ with a size of $|Dom(a_i)| = n_i$. An attribute augmented graph is denoted as $G_a = (V \cup V_a, E \cup E_a)$ where $V_a = \{v_{ij}\}_{i=1, j=1}^{m, \ n_i}$ is the set of attribute vertices and $E_a \subseteq V \times V_a$ is the set of attribute edges. An attribute vertex $v_{ij} \in V_a$ represents that attribute a_i takes the jth value. An attribute edge $(v_i, v_{jk}) \in E_a$ iff $a_j(v_i) = a_{jk}$, i.e., vertex v_i takes the value of a_{jk} on attribute a_j. Accordingly, a vertex $v \in V$ is called a structure vertex and an edge $(v_i, v_j) \in E$ is called a structure edge.

Figure 2 is an attribute augmented graph on the coauthor network example. Two attribute vertices v_{11} and v_{12} representing the topics "XML" and "Skyline" are added. Authors with corresponding topics are connected to the two vertices respectively in dashed lines. We omit the attribute vertices and edges corresponding to the age attribute, for the sake of clear presentation.

3.2 A unified random walk distance

In this paper we use the neighborhood random walk model on the attribute augmented graph G_a to compute a unified distance between vertices in V. The random walk distance between two vertices $v_i, v_j \in V$ is based on the paths consisting of both structure and attribute edges. Thus it effectively combines the structural proximity and attribute similarity of two vertices into one unified measure. The transition probability matrix P_A on G_a is defined as follows.

A structure edge $(v_i, v_j) \in E$ is of a different type from an attribute edge $(v_i, v_{jk}) \in E_a$. The m attributes in Λ may also have different importance. Therefore, they may have different degree of contributions in random walk distance. Without loss of generality, we assume that a structure edge has a weight of ω_0, attribute edges corresponding to $a_1, a_2, \ldots, a_m$ have an edge weight of $\omega_1, \omega_2, \ldots, \omega_m$, respectively. In the following, we will define the transition probabilities between two structure vertices, between a structure vertex and an attribute vertex, and between two attribute vertices. First, the transition probability from a structure vertex v_i to another structure vertex v_j through a structure edge is

$$p_{v_i,v_j} = \begin{cases} \frac{\omega_0}{|N(v_i)|*\omega_0+\omega_1+\cdots+\omega_m}, & if(v_i, v_j) \in E \\ 0, & otherwise \end{cases} \tag{1}$$

where $N(v_i)$ represents the set of structure vertices connected to v_i.

The transition probability from a structure vertex v_i to an attribute vertex v_{jk} through an attribute edge is

$$p_{v_i,v_{jk}} = \begin{cases} \frac{\omega_j}{|N(v_i)|*\omega_0+\omega_1+\cdots+\omega_m}, & if(v_i, v_{jk}) \in E_a \\ 0, & otherwise \end{cases} \tag{2}$$

The transition probability from an attribute vertex v_{ik} to a structure vertex v_j through an attribute edge is

$$p_{v_{ik},v_j} = \begin{cases} \frac{1}{|N(v_{ik})|}, & if(v_{ik}, v_j) \in E_a \\ 0, & otherwise \end{cases} \tag{3}$$

The transition probability between two attribute vertices v_{ip} and v_{jq} is 0 as there is no edge between attribute vertices.

$$p_{v_{ip},v_{jq}} = 0, \forall v_{ip}, v_{jq} \in V_a \tag{4}$$

The transition probability matrix P_A is a $|V \cup V_a| \times |V \cup V_a|$ matrix, where the first $|V|$ rows and columns correspond to the structure vertices and the rest $|V_a|$ rows and columns correspond to the attribute vertices. For the ease of presentation, P_A is represented as

$$P_A = \begin{bmatrix} P_{V_1} & A_1 \\ B_1 & O \end{bmatrix} \tag{5}$$

where P_{V_1} is a $|V| \times |V|$ matrix representing the transition probabilities defined by Eq. 1; A_1 is a $|V| \times |V_a|$ matrix representing the transition probabilities defined by Eq. 2; B_1 is a $|V_a| \times |V|$ matrix representing the transition probabilities defined by Eq. 3; and O is a $|V_a| \times |V_a|$ zero matrix.

Definition 3 (*Random walk distance matrix*) Let P_A be the transition probability matrix of an attribute augmented graph G_a. Given L as the length that a random walk can go, $c \in (0, 1)$ as the random walk restart probability, the unified neighborhood random walk distance matrix R_A is

$$R_A = \sum_{l=1}^{L} c(1-c)^l P_A^l \tag{6}$$

3.3 A review of SA-Cluster

SA-Cluster adopts the *K-Medoids* clustering framework. After initializing the cluster centroids and calculating the random walk distance at the beginning of the clustering process, it repeats the following four steps until convergence.

1. Assign vertices to their closest centroids;
2. Update cluster centroids;
3. Adjust attribute edge weights $\{\omega_1, \ldots, \omega_m\}$;
4. Re-calculate the random walk distance matrix R_A.

Different from traditional K-Medoids, SA-Cluster has two additional steps (i.e., steps 3-4): in each iteration, the attribute edge weights $\{\omega_1, \ldots, \omega_m\}$ are automatically adjusted to reflect the clustering tendencies of different attributes. Interested readers can refer to Zhou et al. (2009) for the proposed mechanism for weight adjustment. According to Eq. 2, when the edge weights $\{\omega_1, \ldots, \omega_m\}$ change, the transition probability matrix P_A changes, so does the neighborhood random walk distance matrix R_A. As a result, the random walk distance matrix has to be re-calculated in each iteration due to the edge weight changes.

Example 1 In Fig. 2, we use ω_1, ω_2 to represent the attribute edge weights of research topic and age, which are initialized to 1.0 at the beginning of clustering. Then we can calculate the transition probability matrix P_A and the random walk distance matrix R_A. Based on R_A, we partition the graph vertices into two clusters where r_1–r_8 belong to one cluster and r_9–r_{11} to the other cluster. We find that in each cluster the attribute values of research topic are much less random than those of age. Many vertices share the same value on research topic in each cluster, which shows that research topic has good clustering tendency. Thus, the weight of research topic ω_1 is increased from 1.0 to 9/7. On the other hand, the vertices within each cluster have different values on age, which shows a quite random distribution. Therefore, the weight of age ω_2 is reduced from 1.0 to 5/7. With the updated weights, the transition probability matrix P_A and the random walk distance matrix R_A are recalculated and the clustering process repeats until convergence.

The cost analysis of SA-Cluster can be expressed as

$$t \cdot (T_{random\_walk} + T_{centroid\_update} + T_{assign})$$

where t is the number of iterations in the clustering process, T_{random_walk} is the cost of computing the random walk distance matrix R_A, $T_{centroid\_update}$ is the cost of updating cluster centroids, and T_{assign} is the cost of assigning all points to cluster centroids.

For $T_{centroid\_update}$ and T_{assign} the time complexity is $O(n)$ where $n = |V|$, since each of these two operations performs a linear scan of the graph vertices. On the other hand, the random walk distance calculation consists of matrix multiplications and additions, according to Eq. 6. Thus the time complexity of T_{random_walk} is $O(L \cdot n_a^3)$ where $n_a = |V \cup V_a|$ is the row or column number of the transition probability matrix P_A. It is clear that T_{random_walk} is the dominant factor in the clustering process. The *repeated calculation* of random walk distance in each iteration can incur a non-trivial efficiency problem for SA-Cluster. We have observed that computing the random walk distance takes 98% of the total clustering time in SA-Cluster.

3.4 Our solution: an incremental approach

The computational bottleneck in the random walk distance computation motivates us to seek alternative solutions with a lower cost. A natural direction to explore is "*can we avoid repeated calculation of random walk distance in the clustering process*?" The goal is to reduce the time of random walk distance calculation. We have observed that the attribute weight adjustments only change the transition probabilities of the attribute edges, but not those of the structure edges. This implies that many elements in the random walk distance matrix may remain unchanged. This property sheds light on the problem: we can design an *incremental calculation* approach to update the random walk distance matrix R_A iteratively. That is, given the original random walk distance matrix R_A and the weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$, efficiently calculate the increment matrix ΔR_A, and then get the updated random walk distance matrix $R_{N,A} = R_A + \Delta R_A$. In this process, we only calculate the *non-zero* elements in ΔR_A, i.e., those elements which are affected by the edge weight changes, but can ignore the unaffected parts of the original matrix. If the number of affected matrix elements is small, this incremental approach will be much more efficient than calculating the full matrix R_A from scratch in each iteration.

However, this incremental approach could be quite challenging, because the boundary between the changed part and the unchanged part of the graph is not clear. The attribute weight adjustments will be propagated to the whole graph in L steps. Let us look at an example first.

Example 2 We select 1,000 authors from database, data mining, artificial intelligence and information retrieval with 3,782 edges for their collaborations. Each author has two attributes: "prolific" and "research topic". The first attribute "prolific" contains two values of "highly prolific" and "low prolific", and the second one "research topic" has 100 different values. Thus the augmented graph contains 1,000 structure vertices and 102 attribute vertices. The attribute edge weights for "prolific" and "research topic" are ω_1, ω_2 respectively. Figure 3 shows three matrices ΔP_A^1, ΔP_A^2 and ΔP_A^{20} corresponding to the increments of the 1st, 2nd, and 20th power of the transition probability matrix, due to the attribute weight increments $\{\Delta\omega_1, \Delta\omega_2\}$. The blue dots represent non-zero elements and the red dashed lines divide each matrix into submatrices according to the block matrix representation in Eq. 5. As shown in Fig. 3, ΔP_A^l becomes denser when l increases.

For ΔP_A^1, the attribute weight increments only affect the transition probabilities in the submatrix A_1, but cause no changes in the other three submatrices. Therefore, most elements in ΔP_A^1 are zero. ΔP_A^2 becomes denser with more non-zero elements.

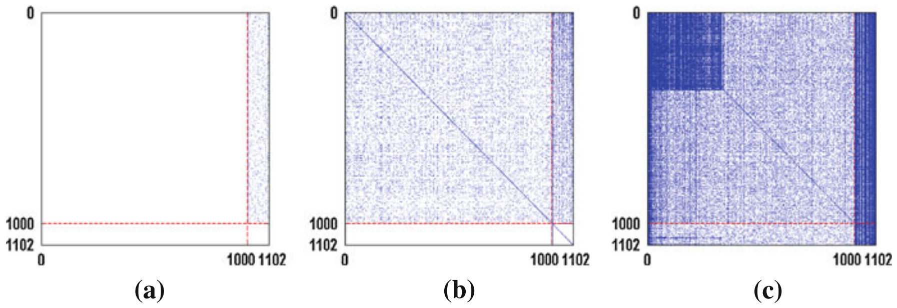


Fig. 3 Matrix increment series. **a** ΔP_A^1, **b** ΔP_A^2, **c** ΔP_A^{20}

ΔP_A^{20} becomes even denser, which demonstrates that the effect of attribute weight increments is propagated to the whole graph through matrix multiplication.

Existing fast random walk (Tong et al. 2006, 2008) or incremental PageRank computation approaches (Desikan 2005; Wu and Raschid 2009) can not be directly applied to our problem. Tong et al. (2006; 2008) proposed an algorithm for fast random walk computation, which relies on partitioning the graph into k clusters apriori, to decompose the transition probability matrix into a within-partition one and a cross-partition one for a lower complexity. However, our graph clustering problem is much more difficult due to the augmented attribute edges and the iterative weight adjustments. The Incremental PageRank Algorithm (IPR) (Desikan et al. 2005) computes PageRank for the evolving Web graph by partitioning the graph into a changed part and an unchanged part. The distribution of PageRank values in the unchanged part will not be affected. Two recent algorithms IdealRank and ApproxRank in Wu and Raschid (2009) compute the PageRank scores in a subgraph, which is a small part of a global graph, by assuming that the scores of external pages, i.e., unchanged pages, are known. Our incremental computation problem is much more challenging than the above problems. As we can see from Fig. 3, although the edge weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$ affect a very small portion of the transition probability matrix P_A, (i.e., see ΔP_A^1), the changes are propagated widely to the whole graph through matrix multiplication (i.e., see ΔP_A^2 and ΔP_A^{20}). It is difficult to partition the graph into a changed part and an unchanged part and focus the computation on the changed part only.

4 The incremental algorithm

In this section, we will describe the incremental algorithm. According to Eq. 6, R_A is the weighted sum of a series of matrices P_A^l, where P_A^l is the l-th power of the transition probability matrix $P_A, l = 1, \ldots, L$. Hence the problem of computing ΔR_A can be decomposed into the subproblems of computing ΔP_A^l for different l values. Therefore, our target is, given the original matrix P_A^l and the edge weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$, compute the increment ΔP_A^l.

4.1 Calculate 1st power matrix increment ΔP_A^1

According to Eq. 5, the transition probability matrix P_A can be expressed as four submatrices P_{V_1}, A_1, B_1 and O. Based on the transition probabilities defined in Eqs. 1–4 and the properties $\sum_{i=1}^{m} \omega_i = m$ and ω_0 is fixed, it is not hard to verify that the attribute weight increments only affect the transition probabilities in the submatrix A_1, but cause no changes in the other three submatrices. Therefore, the increment of the transition probability matrix ΔP_A^1 is denoted as

$$\Delta P_A^1 = \begin{bmatrix} O & \Delta A_1 \\ O & O \end{bmatrix}$$

Consider a probability $p(v_i, v_{jk}) = \frac{\omega_j}{|N(v_i)|*\omega_0+\omega_1+\cdots+\omega_m}$ as defined in Eq. 2. Given a new weight $\omega_j' = \omega_j + \Delta\omega_j$, the probability increment is

$$\begin{aligned}\Delta p(v_i, v_{jk}) &= \frac{\omega_j'}{|N(v_i)| * \omega_0 + \omega_1' + \cdots + \omega_m'} - \frac{\omega_j}{|N(v_i)| * \omega_0 + \omega_1 + \cdots + \omega_m} \\ &= \frac{\Delta\omega_j}{|N(v_i)| * \omega_0 + \omega_1 + \cdots + \omega_m} \\ &= \Delta\omega_j \cdot p(v_i, v_{jk}) \end{aligned} \tag{7}$$

Equation 7 holds because $\omega_j = 1.0$ and $\sum_{i=1}^{m} \omega_i = \sum_{i=1}^{m} \omega_i' = m$. Thus we denote $A_1 = [A_{a_1}, A_{a_2}, \ldots, A_{a_m}]$ where A_{a_i} is a $|V| \times n_i$ matrix representing the transition probabilities from structure vertices in V to attribute vertices corresponding to attribute a_i. The column number n_i corresponds to the n_i possible values in $Dom(a_i)$. An element $A_{a_i}(p, q)$ represents the transition probability from the p-th vertex $v_p \in V$ to the q-th value a_{iq} of a_i. Then according to Eq. 7 ΔA_1 is equal to

$$\Delta A_1 = [\Delta\omega_1 \cdot A_{a_1}, \Delta\omega_2 \cdot A_{a_2}, \ldots, \Delta\omega_m \cdot A_{a_m}] \tag{8}$$

where $\Delta\omega_i \cdot A_{a_i}$ is scalar multiplication, i.e., multiplying every element in A_{a_i} with $\Delta\omega_i$ according to Eq. 7. Then the new transition probability matrix $P_{N,A}$ after the edge weights change is represented as

$$P_{N,A} = \begin{bmatrix} P_{V_1} & A_1 + \Delta A_1 \\ B_1 & O \end{bmatrix} = \begin{bmatrix} P_{V_1} & A_{N,1} \\ B_1 & O \end{bmatrix}$$

4.2 Calculate l-th power matrix increment ΔP_A^l

Similar to the computation of ΔP_A^1, we can calculate ΔP_A^l ($l \geq 2$), with a more complicated computation. The original l-th power matrix $P_A^l = P_A^{l-1} \times P_A$ is represented as

$$\begin{aligned} P_A^l &= \begin{bmatrix} P_{V_{l-1}} & A_{l-1} \\ B_{l-1} & C_{l-1} \end{bmatrix} \times \begin{bmatrix} P_{V_1} & A_1 \\ B_1 & O \end{bmatrix} \\ &= \begin{bmatrix} P_{V_{l-1}} P_{V_1} + A_{l-1} B_1 & P_{V_{l-1}} A_1 \\ B_{l-1} P_{V_1} + C_{l-1} B_1 & B_{l-1} A_1 \end{bmatrix} \end{aligned}$$

Similarly, the new matrix $P_{N,A}^l = P_{N,A}^{l-1} \times P_{N,A}$ given the weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$ is

$$\begin{aligned} P_{N,A}^l &= \begin{bmatrix} P_{N,V_{l-1}} & A_{N,l-1} \\ B_{N,l-1} & C_{N,l-1} \end{bmatrix} \times \begin{bmatrix} P_{V_1} & A_{N,1} \\ B_1 & O \end{bmatrix} \\ &= \begin{bmatrix} P_{N,V_{l-1}} P_{V_1} + A_{N,l-1} B_1 & P_{N,V_{l-1}} A_{N,1} \\ B_{N,l-1} P_{V_1} + C_{N,l-1} B_1 & B_{N,l-1} A_{N,1} \end{bmatrix} \end{aligned}$$

Then the l-th power transition probability matrix increment ΔP_A^l is denoted as

$$\Delta P_A^l = \begin{bmatrix} \Delta P_{V_l} & \Delta A_l \\ \Delta B_l & \Delta C_l \end{bmatrix}$$

Based on the original matrix P_A^l and the new matrix $P_{N,A}^l$, the increment ΔP_{V_l} is

$$\begin{aligned} \Delta P_{V_l} &= (P_{N,V_{l-1}} P_{V_1} + A_{N,l-1} B_1) - (P_{V_{l-1}} P_{V_1} + A_{l-1} B_1) \\ &= (P_{V_{l-1}} + \Delta P_{V_{l-1}}) P_{V_1} + (A_{l-1} + \Delta A_{l-1}) B_1 - (P_{V_{l-1}} P_{V_1} + A_{l-1} B_1) \\ &= \Delta P_{V_{l-1}} P_{V_1} + \Delta A_{l-1} B_1 \end{aligned}$$

The increment ΔB_l is

$$\begin{aligned} \Delta B_l &= (B_{N,l-1} P_{V_1} + C_{N,l-1} B_1) - (B_{l-1} P_{V_1} + C_{l-1} B_1) \\ &= (B_{l-1} + \Delta B_{l-1}) P_{V_1} + (C_{l-1} + \Delta C_{l-1}) B_1 - (B_{l-1} P_{V_1} + C_{l-1} B_1) \\ &= \Delta B_{l-1} P_{V_1} + \Delta C_{l-1} B_1 \end{aligned}$$

The increment ΔA_l is

$$\begin{aligned} \Delta A_l &= P_{N,V_{l-1}} A_{N,1} - P_{V_{l-1}} A_1 \\ &= (P_{V_{l-1}} + \Delta P_{V_{l-1}})(A_1 + \Delta A_1) - P_{V_{l-1}} A_1 \\ &= P_{V_{l-1}} \Delta A_1 + \Delta P_{V_{l-1}} A_{N,1} \end{aligned} \tag{9}$$

In Eq. 9, there is one component $P_{V_{l-1}} \Delta A_1$. As shown in Eq. 8, $\Delta A_1 = [\Delta\omega_1 \cdot A_{a_1}, \ldots, \Delta\omega_m \cdot A_{a_m}]$, we then have

$$\begin{aligned} P_{V_{l-1}} \Delta A_1 &= P_{V_{l-1}} [\Delta\omega_1 \cdot A_{a_1}, \ldots, \Delta\omega_m \cdot A_{a_m}] \\ &= [\Delta\omega_1 \cdot P_{V_{l-1}} A_{a_1}, \ldots, \Delta\omega_m \cdot P_{V_{l-1}} A_{a_m}] \end{aligned}$$

Note that the submatrix A_l in P_A^l is computed by the following submatrix multiplication:

$$A_l = P_{V_{l-1}} A_1 + A_{l-1} O = P_{V_{l-1}} A_1$$

If we rewrite A_l as a series of $|V| \times n_i$ submatrices as $A_l = [A_{l,a_1}, A_{l,a_2}, \ldots, A_{l,a_m}]$, then $A_{l,a_i} = P_{V_{l-1}} A_{a_i}$. As a result, $P_{V_{l-1}} \Delta A_1$ can be expressed as

$$\begin{aligned} P_{V_{l-1}} \Delta A_1 &= [\Delta\omega_1 \cdot P_{V_{l-1}} A_{a_1}, \ldots, \Delta\omega_m \cdot P_{V_{l-1}} A_{a_m}] \\ &= [\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}] \end{aligned}$$

Therefore, to compute $P_{V_{l-1}} \Delta A_1$ in Eq. 9, we only need to compute $[\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}]$. The advantage is that $\Delta\omega_i \cdot A_{l,a_i}$ is scalar multiplication, which is much cheaper than the matrix multiplication on $P_{V_{l-1}} \Delta A_1$. Combining the above equations, we have

$$\Delta A_l = [\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}] + \Delta P_{V_{l-1}} A_{N,1} \tag{10}$$

where the first part represents the attribute increment (i.e., the weight increments $\Delta\omega_i$'s on A_l), while the second part represents the accumulative increment from $\Delta P_{V_{l-1}}$.

Similarly, the increment ΔC_l is

$$\begin{aligned} \Delta C_l &= B_{N,l-1} A_{N,1} - B_{l-1} A_1 \\ &= (B_{l-1} + \Delta B_{l-1})(A_1 + \Delta A_1) - B_{l-1} A_1 \\ &= B_{l-1} \Delta A_1 + \Delta B_{l-1} A_{N,1} \\ &= [\Delta\omega_1 \cdot C_{l,a_1}, \ldots, \Delta\omega_m \cdot C_{l,a_m}] + \Delta B_{l-1} A_{N,1} \end{aligned}$$

where we represent $C_l = [C_{l,a_1}, C_{l,a_2}, \ldots, C_{l,a_m}]$.

In summary, the l-th power matrix increment ΔP_A^l can be calculated based on: (1) the original transition probability matrix P_A and increment matrix ΔA_1, (2) the (l-1)-th power matrix increment ΔP_A^{l-1}, and (3) the original l-th power submatrices A_l and C_l. The key is that, if ΔA_1 and ΔP_A^{l-1} contain many zero elements, we can apply sparse matrix representation to speed up the matrix multiplication.

4.3 The incremental algorithm

Algorithm 1 presents the incremental algorithm for calculating the new random walk distance matrix $R_{N,A}$ given the original R_A and the weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$. The algorithm iteratively computes the increments ΔP_A^l for $l = 1, \ldots, L$, and accumulates them into the increment matrix ΔR_A according to Eq. 6. Finally the new random walk distance matrix $R_{N,A} = R_A + \Delta R_A$ is returned.

The total runtime cost of the clustering process with Inc-Cluster can be expressed as

$$T_{random\_walk} + (t-1) \cdot T_{inc} + t \cdot (T_{centroid\_update} + T_{assign}) \tag{11}$$

Algorithm 1 The Incremental Algorithm Inc-Cluster

```
Input: The original matrices $R_A, P_A, A_l, C_l, l = 2, \ldots, L$
        the attribute edge weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$
Output: The new random walk distance matrix $R_{N,A}$

1: Calculate $\Delta P_A^1$ according to Eq. 8;
2: $\Delta R_A = c(1-c)\Delta P_A^1$;
3: for $l = 2, \ldots, L$
4:     $\Delta P_{V_l} = \Delta P_{V_{l-1}} P_{V_1} + \Delta A_{l-1} B_1$;
5:     $\Delta B_l = \Delta B_{l-1} P_{V_1} + \Delta C_{l-1} B_1$;
6:     $\Delta A_l = [\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}] + \Delta P_{V_{l-1}} A_{N,1}$;
7:     $\Delta C_l = [\Delta\omega_1 \cdot C_{l,a_1}, \ldots, \Delta\omega_m \cdot C_{l,a_m}] + \Delta B_{l-1} A_{N,1}$;
8:     $\Delta R_A += c(1-c)^l \Delta P_A^l$;
9: end for
10: return $R_{N,A} = R_A + \Delta R_A$;
```

where T_{inc} is the time for incremental computation and T_{random_walk} is the time for computing the random walk distance matrix at the beginning of clustering. The speedup ratio r between SA-Cluster and Inc-Cluster is

$$r = \frac{t(T_{random\_walk} + T_{centroid\_update} + T_{assign})}{T_{random\_walk} + (t-1)T_{inc} + t(T_{centroid\_update} + T_{assign})}$$

Since $T_{inc}, T_{centroid\_update}, T_{assign} \ll T_{random\_walk}$, the speedup ratio is approximately

$$r \approx \frac{t \cdot T_{random\_walk}}{T_{random\_walk}} = t$$

Therefore, Inc-Cluster can improve the runtime cost of SA-Cluster by approximately t times, where t is the number of iterations in clustering.

In the following, we further analyze the overall time complexity of Inc-Cluster. According to Algorithm 1, the dominant factor in the incremental computation is $\Delta P_{V_{l-1}} P_{V_1}$, the multiplication of two $n \times n$ matrices, where $n = |V|$. The other matrix multiplications are much cheaper, when we assume $|V_a| \ll |V|$. As $\Delta P_{V_{l-1}}$ is a sparse matrix, the matrix multiplication $\Delta P_{V_{l-1}} P_{V_1}$ takes $O(mn)$ time where m is the number of nonzero elements in $\Delta P_{V_{l-1}}$. Therefore, the time complexity of T_{inc} is $O(Lmn)$. In contrast, according to Sect. 3, the time complexity of T_{random_walk} is $O(L \cdot n_a^3)$ where $n_a = |V \cup V_a|$ is the row or column number of P_A. Therefore, according to Eq. 11, the overall time complexity of Inc-Cluster is $O(Ln_a^3 + (t-1)Lmn)$, which is much more efficient than the complexity of SA-Cluster as $O(tLn_a^3)$.

4.4 Parallel matrix computation

The Inc-Cluster algorithm can achieve further speedup with the parallel matrix computation techniques on a multicore architecture. In Algorithm 1, we divide the increment matrix ΔP_A^l into four submatrices ΔP_{V_l}, ΔA_l, ΔB_l and ΔC_l. The computation of the four submatrices can be assigned to different cores in parallel. Among the four submatrices, we observed that the major cost is from the computation of ΔP_{V_l}, as its size is $|V| \times |V|$, which is much larger than the sizes of the other three. Therefore, we want to recursively partition this matrix for parallel multiplication. In the literature, there are mainly two proposed fast matrix block multiplication algorithms: a group-theoretic based approach by Cohn et al. (2005) and the Strassen algorithm by Strassen (1969). In particular, the Strassen algorithm partitions matrices into equal-sized four block matrices and performs block multiplications by reducing the number of multiplications to 7. In our current implementation, we follow the Strassen idea and recursively partition the matrix into blocks for parallel computation.

5 Complexity analysis

In this section, we will perform some complexity analysis to estimate the number of zero elements in ΔP_A^l, as an indication to show how much cost Inc-Cluster can save. Intuitively, the more zero elements in the matrix increment ΔP_A^l, the less cost the incremental algorithm has. It is hard to give a closed form analytical result for a general $l \in \{1, \ldots, L\}$, because we need to consider all possible length-l paths between any two vertices. So we focus on the analysis on ΔP_A^2. Given a general attributed graph, we will provide an upper bound and a lower bound of the number of zero elements in ΔP_A^2. This quantity directly affects the computational complexity of the incremental calculation.

Although we cannot provide the theoretical bounds for ΔP_A^l $(l > 2)$, we observe in Fig. 3 that the number of non-zero elements increases as l increases. However, we also observe from experiments, a large number of entries in ΔP_A^l approach to zero quickly when l increases, due to the multiplication of probabilities on the sequence of edges. Confirmed by our testing, over 75% entries in ΔP_A^l become smaller than a very small threshold and can be treated as zero. Therefore, practically the number of non-zero elements in ΔP_A^l is very small even for large l values.

In our analysis, we use the following notations: the m attributes $a_1, \ldots, a_m$ contain $n_1, \ldots, n_m$ values respectively. The number of structure vertices is $|V| = n$. Note that the following derived bounds do not make any assumption about the type of data or the value of m.

5.1 Upper bound of the number of zero elements in ΔP_A^2

Lemma 1 *There are totally $\prod_{i=1}^{m} n_i$ combinations of attribute values among the m attributes, since an attribute a_i takes n_i values. Assume each combination has at least one vertex (without this assumption, we can find a special case with a trivial upper* *bound). When all vertices are evenly distributed in the $\prod_{i=1}^{m} n_i$ combinations of attribute values, i.e., each combination has $\frac{n}{\prod_{i=1}^{m} n_i}$ vertices, it gives the upper bound of the number of zero elements in ΔP_A^2.*

Proof See Appendix 9.1. □

Theorem 1 *The upper bound of the number of zero elements in ΔP_A^2 is*

$$\frac{n^2 \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i} \tag{12}$$

Proof If two vertices $v_i, v_j \in V$ have no common values on any attributes, then they are not connected to any common attribute vertices, thus $\Delta P_A^2(i, j) = \Delta P_A^2(j, i) = 0$. For one combination of the attribute values, there are $\prod_{i=1}^{m}(n_i - 1)$ combinations which do not share any attribute values with this combination. Since all vertices are evenly distributed in the $\prod_{i=1}^{m} n_i$ combinations of attribute values, there are $\frac{n}{\prod_{i=1}^{m} n_i}$ vertices belonging to each combination. Therefore, for any vertex v_i, the total number of vertices which do not share any attribute values with v_i is $\frac{n \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$. Accordingly, there are $\frac{n \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$ zero elements in $\Delta P_A^2(i, :)$. Since there are totally n vertices in the graph, the total number of zero elements in ΔP_A^2 is

$$\frac{n^2 \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$$

□

$\frac{n^2 \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$ is in the scale of $O(n^2)$, which implies that most elements in ΔP_A^2 do not change. This corresponds to the best case of the incremental computation, since only a small number of elements in ΔP_A^2 need to be updated.

5.2 Lower bound of the number of zero elements in ΔP_A^2

Lemma 2 *Assume each attribute value combination has at least one vertex. Among the $\prod_{i=1}^{m} n_i$ combinations of attribute values, assume for each of the first $\prod_{i=1}^{m} n_i - 1$ combinations, there exists exactly one vertex with the attribute values corresponding to that combination. The remaining $n - (\prod_{i=1}^{m} n_i - 1)$ vertices have the same attribute values corresponding to the last combination. This case gives the lower bound of the number of zero elements in ΔP_A^2.*

Proof See Appendix 9.2. □

Theorem 2 *The lower bound of the number of zero elements in ΔP_A^2 is*

$$\left(2n - \prod_{i=1}^{m} n_i\right) \times \prod_{i=1}^{m}(n_i - 1) \tag{13}$$

Proof Without loss of generality, we assume that exactly one vertex belongs to each of the first $\prod_{i=1}^{m} n_i - 1$ combinations of the attribute values. The set of such vertices is denoted as S. The set of the remaining vertices belonging to the last combination of attribute values is denoted as T. Let $S = S_1 \cup S_2$ where S_1 is the set of vertices which do not share any attribute values with vertices in T; S_2 is the set of vertices which share one or more attribute values with vertices in T.

There are three cases to be discussed in the following to count the number of zero elements in ΔP_A^2.

Case 1 Consider two vertices u and v. If u, v do not share the value on an attribute a_i, then v can take any of the other $n_i - 1$ values except the value taken by u. Since vertices in T and S_1 do not share any attribute values on the m attributes, there are totally $\prod_{i=1}^{m}(n_i - 1)$ combinations of attribute values that do not share with vertices in T. As we have assumed that there is exactly one vertex for each of such combinations, the size of S_1 is $|S_1| = \prod_{i=1}^{m}(n_i - 1)$ and the size of T is $|T| = n - (\prod_{i=1}^{m} n_i - 1)$. If two vertices v_i, v_j have no common values on any attributes, then $\Delta P_A^2(i, j) = \Delta P_A^2(j, i) = 0$. Therefore, $\forall v_i \in T, \forall v_j \in S_1$, $\Delta P_A^2(i, j) = 0$ and $\Delta P_A^2(j, i) = 0$. The number of such elements between T and S_1 is

$$LB_1 = 2|T| \times |S_1| = 2\left(n - \left(\prod_{i=1}^{m} n_i - 1\right)\right) \times \prod_{i=1}^{m}(n_i - 1) \tag{14}$$

Case 2 There exist some vertices in S which do not share any attribute values with any vertex in S_1. We denote this set as S_0, $S_0 \subset S$. The size of S_0 is $|S_0| = \prod_{i=1}^{m}(n_i - 1) - 1$. So the total number of zero elements in ΔP_A^2 is

$$2|S_1| \times |S_0| = 2\prod_{i=1}^{m}(n_i - 1) \times \left(\prod_{i=1}^{m}(n_i - 1) - 1\right)$$

Since $S_0 \cap S_1 \neq \emptyset$, the above number double counts the following case: $v_i, v_j \in S_1$ and v_i, v_j do not share any attribute values. As a result, we have to deduct from the above $\prod_{i=1}^{m}(n_i - 1) \times \prod_{i=1}^{m}(n_i - 2)$ elements. Finally the number of zero elements in ΔP_A^2 in case 2 is

$$LB_2 = 2\prod_{i=1}^{m}(n_i - 1) \times \left(\prod_{i=1}^{m}(n_i - 1) - 1\right) - \prod_{i=1}^{m}(n_i - 1) \times \prod_{i=1}^{m}(n_i - 2) \tag{15}$$

Case 3 There exist some vertices in S which do not share any attribute values with those in S_2. The size of S_2 is $|S_2| = \prod_{i=1}^{m} n_i - 1 - \prod_{i=1}^{m}(n_i - 1)$. So the total number of zero elements in ΔP_A^2 in case 3 is

$$\left(\prod_{i=1}^{m} n_i - 1 - \prod_{i=1}^{m}(n_i - 1)\right) \times \prod_{i=1}^{m}(n_i - 1)$$

However, the elements between any $v_i \in S_1$, $v_j \in S_2$ have been counted in case 2. So we should deduct the repeated counts. For a vertex $v_i \in S_1$, there are $\prod_{i=1}^{m}(n_i-1) - \prod_{i=1}^{m}(n_i-2) - 1$ vertices in S_2 which do not share any attribute values with it. Thus the number of repeated counts is $(\prod_{i=1}^{m}(n_i-1) - \prod_{i=1}^{m}(n_i-2) - 1) \times \prod_{i=1}^{m}(n_i-1)$. Finally the number of zero elements for case 3 is

$$LB_3 = \left(\prod_{i=1}^{m} n_i + \prod_{i=1}^{m}(n_i-2) - 2\prod_{i=1}^{m}(n_i-1)\right) \times \prod_{i=1}^{m}(n_i-1) \tag{16}$$

By adding up LB_1, LB_2 and LB_3, we can generate the lower bound of the number of zero elements in ΔP_A^2.

$$LB = LB_1 + LB_2 + LB_3 = \left(2n - \prod_{i=1}^{m} n_i\right) \times \prod_{i=1}^{m}(n_i-1)$$

□

As m and n_i, $i = 1, \ldots, m$, are usually much smaller than n, LB is in the scale of $O(n)$, which is $\ll n^2$, the number of elements in ΔP_A^2. Thus the lower bound corresponds to the worst case of the incremental computation, since most elements in ΔP_A^2 need to be updated.

6 Storage cost analysis

In this section, we will analyze the storage cost of the incremental algorithm Inc-Cluster. Then we will discuss some techniques to further save space.

6.1 The storage cost

According to Algorithm 1, we need to store a series of submatrices, as listed in the following.

– The original transition probability matrix P_A. We need to use P_{V_1}, B_1, ΔA_1 and $A_{N,1}$ for incremental computation. According to Eq. 8, $\Delta A_1 = [\Delta\omega_1 \cdot A_{a_1}, \ldots, \Delta\omega_m \cdot A_{a_m}]$ where $A_1 = [A_{a_1}, A_{a_2}, \ldots, A_{a_m}]$. In addition $A_{N,1} = A_1 + \Delta A_1$. Therefore, ΔA_1 and $A_{N,1}$ can be derived from A_1 with some simple computation. In summary, it is enough to store the transition probability matrix P_A.

– The $(l-1)$-th power matrix increment ΔP_A^{l-1}. To calculate the l-th power matrix increment ΔP_A^l, we need to use $\Delta P_{V_{l-1}}$, ΔA_{l-1}, ΔB_{l-1} and ΔC_{l-1}. Therefore, we need to store the $(l-1)$-th power matrix increment ΔP_A^{l-1}. After ΔP_A^l is computed, we can discard ΔP_A^{l-1} and save ΔP_A^l in turn for the computation of ΔP_A^{l+1} in the next iteration.

– A series of A_l and C_l for $l = 2, \ldots, L$. In Eq. 10, we have derived $P_{V_{l-1}} \Delta A_1 = [\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}]$. The scalar multiplication $\Delta\omega_i \cdot A_{l,a_i}$ is cheaper than the matrix multiplication $P_{V_{l-1}} \Delta A_1$. In addition, this is more space efficient because we only need to store A_l, but not $P_{V_{l-1}}$. The size of A_l is $|V| \times |V_a| = n \times \sum_{i=1}^m n_i$, which is much smaller than the size of $P_{V_{l-1}}$ as $|V| \times |V| = n^2$, because usually $\sum_{i=1}^m n_i \ll n$. For example, in our experiments, we test a DBLP network with $n = 84,170$ and $\sum_{i=1}^m n_i = 103$. Similarly, to compute $B_{l-1} \Delta A_1 = [\Delta\omega_1 \cdot C_{l,a_1}, \ldots, \Delta\omega_m \cdot C_{l,a_m}]$, we only need to store C_l, but not B_{l-1}. The size of C_l is $|V_a| \times |V_a| = (\sum_{i=1}^m n_i)^2$, which is much smaller than the size of B_{l-1} as $|V_a| \times |V| = \sum_{i=1}^m n_i \times n$.

Total storage cost. Considering the above factors, the total space cost of Inc-Cluster is

$$\begin{aligned} T_{total} &= size(R_A) + size(P_A) + size(\Delta P_A^{l-1}) + size(\Delta P_A^l) \\ &\quad + \sum_{l=2}^{L} size(A_l) + \sum_{l=2}^{L} size(C_l) \\ &= |V|^2 + 3(|V| + |V_a|)^2 + (L-1)(|V| \times |V_a| + |V_a|^2) \\ &= n^2 + 3\left(n + \sum_{i=1}^m n_i\right)^2 + (L-1)\left(n \cdot \sum_{i=1}^m n_i + \left(\sum_{i=1}^m n_i\right)^2\right) \end{aligned}$$

On the other hand, the non-incremental clustering algorithm SA-Cluster has to store four matrices in memory including P_A, P_A^{l-1}, P_A^l and R_A. So the extra space used by Inc-Cluster compared with SA-Cluster is

$$T_{extra} = (L-1)\left(n \cdot \sum_{i=1}^m n_i + \left(\sum_{i=1}^m n_i\right)^2\right) \tag{17}$$

which is linear of n. Therefore, Inc-Cluster uses a small amount of extra space compared with SA-Cluster.

6.2 Sparse matrix and matrix pruning

We further use sparse matrix representation and matrix pruning to reduce the storage cost.

We have observed that many graphs, for example, coauthor networks, are usually very sparse. The number of edges is much smaller than the possible number of edges in a complete graph. As a result, there are many zeros in the original transition probability matrix. Similarly, the increment matrices ΔP_A^l, $l = 1, \ldots, L$, may also contain many zero elements. In this case, we can use the sparse matrix representation, i.e., we store the indices and values of the non-zero elements. The computational complexity of sparse matrix operation is proportional to the number of nonzero elements in the matrix.

Although the initial transition probability matrix P_A and the lower order increment matrices could be quite sparse, the higher order increment matrices ΔP_A^l may not be sparse, as shown in Fig. 3c. This is due to the propagated changes when the random walk takes more steps. In this case, the sparse matrix representation is less useful. However we have observed that among the non-zero elements, many have very small values close to 0. Therefore, for the increment matrices ΔP_A^l, we perform matrix pruning to remove those elements whose values are no greater than a threshold δ. After the pruning, we could use the sparse matrix representation to store the non-zero elements.

7 Experimental study

In this section, we performed extensive experiments to evaluate the performance of Inc-Cluster on real graph data. All experiments were done in Matlab on a Dell PowerEdge R900 server with four 2.67GHz CPUs and 128GB main memory running Windows Server 2008.

7.1 Experimental datasets

We use the DBLP Bibliography data with 10,000 authors from four research areas of database, data mining, information retrieval and artificial intelligence. We build a coauthor graph where nodes represent authors and edges represent their coauthor relationships. In addition, we use two attributes: *prolific* and *primary topic*. For "prolific", authors with $\geq$20 papers are labeled as highly prolific; authors with $\geq$10 and $<$20 papers are labeled as prolific and authors with $<$10 papers are labeled as low prolific. For "primary topic", we use a topic modeling approach by Hofmann (1999) to extract 100 topics from a document collection composed of paper titles from the selected authors. Each extracted topic consists of a probability distribution of keywords which are most representative of the topic. Then each author will have one out of 100 topics as his/her primary topic.

We also use a larger DBLP dataset with 84,170 authors, selected from the following areas: database, data mining, information retrieval, machine learning, artificial intelligence, computer systems, theory, computer vision, architecture, programming language, networking, simulation, natural language processing, multimedia, and human-computer interaction. The coauthor graph and the vertex attributes are defined similarly as in the 10,000 coauthor network.

7.2 Comparison methods and evaluation

We tested the following algorithms for the clustering quality and efficiency comparison.

– **Inc-Cluster** Our proposed algorithm which incrementally updates the random walk distance matrix.
– **SA-Cluster** The non-incremental graph clustering algorithm (Zhou et al. 2009) which considers both structural and attribute similarities.

- **S-Cluster** The graph clustering algorithm which only considers topological structure. Random walk distance is used to measure vertex closeness while attribute similarity is ignored.
- **W-Cluster** A fictitious clustering algorithm which combines structural and attribute similarities through a weighted function as $\alpha \cdot d_S(v_i, v_j) + \beta \cdot d_A(v_i, v_j)$, where $d_S(v_i, v_j)$ is the random walk distance, and $d_A(v_i, v_j)$ is their attribute similarity, and the weighting factors are $\alpha = \beta = 0.5$.
- **K-SNAP** The K-SNAP algorithm (Tian et al. 2008) that groups vertices with the same attribute values into one cluster.

Evaluation measures We use the measures of *density* and *entropy* to evaluate the quality of clusters $\{V_i\}_{i=1}^k$ generated by different methods. The definitions are as follows.

$$density\left(\{V_i\}_{i=1}^k\right) = \sum_{i=1}^{k} \frac{|\{(v_p, v_q)|v_p, v_q \in V_i, (v_p, v_q) \in E\}|}{|E|}$$

$$entropy\left(\{V_i\}_{i=1}^k\right) = \sum_{i=1}^{m} \frac{\omega_i}{\sum_{p=1}^{m} \omega_p} \sum_{j=1}^{k} \frac{|V_j|}{|V|} entropy(a_i, V_j)$$

where $entropy(a_i, V_j) = -\sum_{n=1}^{n_i} p_{ijn} \log_2 p_{ijn}$ and p_{ijn} is the percentage of vertices in cluster j which have value a_{in} on attribute a_i. $entropy(\{V_i\}_{i=1}^k)$ measures the weighted entropy from all attributes over k clusters.

Besides the clustering quality comparison, we also compare the efficiency of these methods.

7.3 Clustering quality comparison

Since SA-Cluster and Inc-Cluster generate the same clustering results, their quality results are shown in the same column in Figs. 4 and 5.

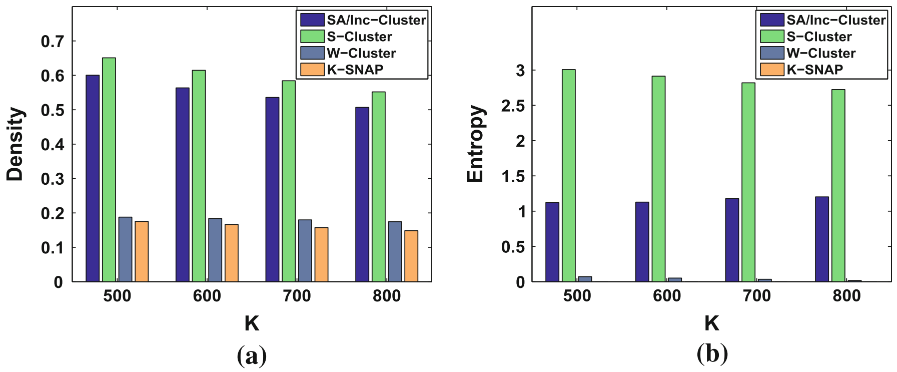


Fig. 4 Cluster quality on DBLP 10,000 authors. **a** density, **b** entropy

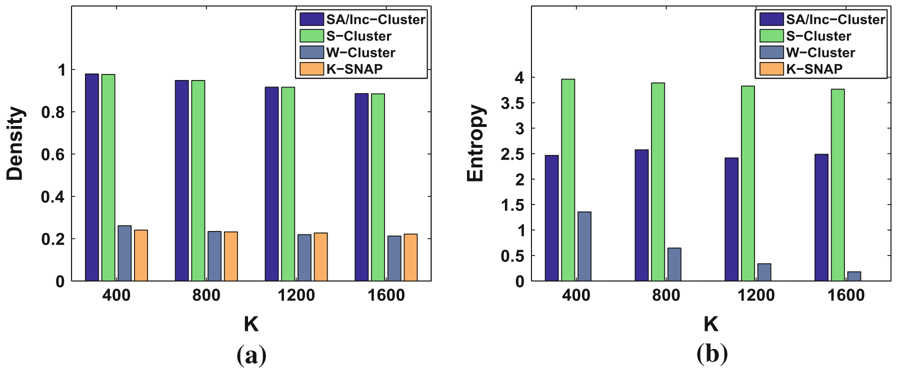


Fig. 5 Cluster quality on DBLP 84,170 authors. **a** density, **b** entropy

Figure 4a shows the density on the DBLP graph with 10,000 authors by different methods. The density values by SA-Cluster and Inc-Cluster are around 0.51–0.60, which are slightly lower than those of S-Cluster. The density values by W-Cluster and K-SNAP are much lower, in the range of 0.15–0.18. This shows the clusters generated by W-Cluster and K-SNAP have a very loose intra-cluster structure.

Figure 4b shows the entropy comparison on DBLP with 10,000 authors. S-Cluster has the highest entropy around 2.7–3.0, because it partitions a graph without considering vertex attributes. SA-Cluster and Inc-Cluster have a low entropy around 1.1–1.2. W-Cluster has an even lower entropy but also a very low density. This is because its distance function combines structural and attribute similarities through a weighted function. However, as it is not clear how to set or tune the weighting factors α and β, it is hard to achieve an optimal result on W-Cluster. Since K-SNAP strictly enforces the attribute homogeneity in each cluster, K-SNAP achieves an entropy of 0.

Figure 5a, b show the density and entropy on DBLP with 84,170 authors. These two figures have a similar trend with Fig. 4a, b. SA-Cluster and Inc-Cluster achieve similar high density values (above 0.90) with S-Cluster, but with much lower entropy. W-Cluster and K-SNAP achieve very low entropy (the entropy by K-SNAP is 0), but with very low density values at 0.2–0.3. The comparison on both density and entropy demonstrates that both SA-Cluster and Inc-Cluster achieve a very good balance between the structural cohesiveness and attribute homogeneity.

7.4 Clustering efficiency comparison

In this experiment, we compare the efficiency of different clustering algorithms. Figure 6a, b show the clustering time on DBLP with 10,000 and 84,170 authors respectively. We make the following observations on the runtime costs of different methods. First, SA-Cluster is about 2–4.3 times slower than Inc-Cluster, as it iteratively computes the random walk distance matrix from scratch. It takes 24 s for the random walk computation but only 4.6 s for the incremental update on the DBLP 10,000 author dataset; and it takes 122 s for the random walk but only 18.4 seconds for the incremental update on the DBLP 84,170 author dataset. Thus, T_{random_walk} is about 5.2–6.6 times slower

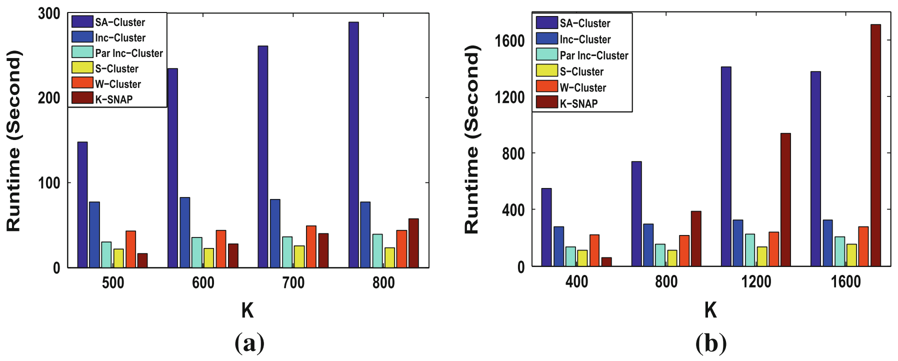
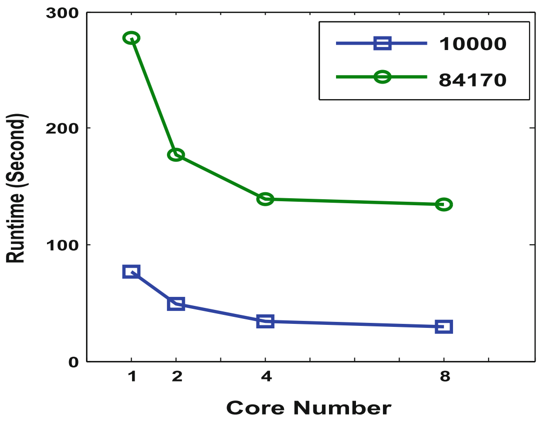


Fig. 6 Clustering efficiency. **a** 10,000 authors, **b** 84,170 authors

Fig. 7 Parallel Inc-Cluster runtime versus core number



than T_{inc}. Second, parallel Inc-Cluster (denoted as Par Inc-Cluster) with 8 cores can reduce the time of Inc-Cluster by 52–62%, which demonstrates the effectiveness of the parallel matrix multiplication. Third, the runtime of S-Cluster and W-Cluster is in the same scale with Inc-Cluster, but the runtime of K-SNAP increases dramatically with the cluster number k.

Figure 7 shows the effectiveness of parallel matrix computation in Inc-Cluster with different number of cores. On both the DBLP datasets with 10,000 and 84,170 authors, we can see that using more cores gradually decreases the runtime of Inc-Cluster. When 8 cores are used for parallel matrix multiplication, the total runtime is reduced by 52–62%, compared with the single core case.

The statistics on the number of zero elements in ΔP_A^2 also verify our previously proved bounds. On DBLP 10, 000 author dataset, there are 24M zero entries in ΔP_A^2, while the theoretical upper and lower bounds are 66M and 4M, respectively. On DBLP 84, 170 author dataset, there are 4.4B zero entries, while the upper and lower bounds are 4.7B and 33M, respectively. Note here the upper bound and lower bound refer to the number of zero entries in the best case and the worst case. The actual number of zero entries depends on the attribute value distribution on the vertices and should be within the lower and upper bounds.

8 Conclusion

In this paper, we propose an incremental algorithm Inc-Cluster to quickly compute a random walk distance matrix, in the context of graph clustering considering both structural and attribute similarities. To avoid recalculating the random walk distances from scratch in each iteration due to the attribute weight changes, we divide the transition probability matrix into submatrices and incrementally update each one. Time complexity analysis is provided to show the properties of Inc-Cluster. Experimental results show that Inc-Cluster achieves significant speedup over SA-Cluster, while achieving the same clustering quality.

Acknowledgments The work described in this paper was supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project no.: CUHK 419008, 411310 and 411211).

9 Appendix

9.1 Proof of Lemma 1

We will prove Lemma 1, which says when all vertices are evenly distributed in the $\prod_{i=1}^{m} n_i$ combinations of attribute values, it gives an upper bound of the number of zero elements in $\varDelta P_A^2$.

Proof We denote as $\varPhi$ the even distribution of graph vertices to the $\prod_{i=1}^{m} n_i$ combinations of attribute values. We can generate a new vertex distribution, $\varGamma$, by moving an arbitrary vertex $v \in V$ from one arbitrary attribute value combination (denoted as η) in $\varPhi$ to another (denoted as ξ).

If η and ξ contain the same values in some or all attributes, the combinations of attribute values which do not share any attribute values with η will not include ξ, and vice versa. Then the set of graph vertices which have no common attribute values with v remains the same. Hence the number of zero elements in $\varDelta P_A^2$ is the same as that of $\varPhi$. Therefore, in this situation, $\varGamma$ is equivalent to $\varPhi$, i.e., both give the upper bound of the number of zero elements in $\varDelta P_A^2$ as $\frac{n^2 \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$.

If η and ξ do not share any attribute values, the combinations of attribute values which do not share any attribute values with η will include ξ, and vice versa. Then, the number of combinations of attribute values which do not share any attribute values with v is $\prod_{i=1}^{m}(n_i - 1)$, which include ξ plus the other $\prod_{i=1}^{m}(n_i - 1) - 1$ combinations before the movement, or η plus the other $\prod_{i=1}^{m}(n_i - 1) - 1$ combinations after the movement. The vertices corresponding to the other $\prod_{i=1}^{m}(n_i - 1) - 1$ combinations remain unchanged. However, except v itself, the number of vertices which do not share any attribute values with v in ξ or η is reduced from $\frac{n}{\prod_{i=1}^{m} n_i}$ to $\frac{n}{\prod_{i=1}^{m} n_i} - 1$ due to the movement of v from η to ξ. Hence, the number of zero elements in ΔP_A^2 is reduced to $\frac{n^2 \times \prod_{i=1}^m (n_i-1)}{\prod_{i=1}^m n_i} - 2$, which is less than the upper bound $\frac{n^2 \times \prod_{i=1}^m (n_i-1)}{\prod_{i=1}^m n_i}$.

Based on the above analysis, we demonstrate that Φ is the vertex distribution that gives the upper bound of the number of zero elements in ΔP_A^2. □

9.2 Proof of Lemma 2

We will prove Lemma 2, which says for each of the first $\prod_{i=1}^m n_i - 1$ combinations, there exists exactly one vertex with the attribute vector corresponding to that combination. The remaining $n - (\prod_{i=1}^m n_i - 1)$ vertices have the same attribute vector corresponding to the last combination. Then this case gives a lower bound of the number of zero elements in ΔP_A^2.

Proof We denote as Ψ the graph vertex distribution described in Lemma 2. Recall S denotes the set of vertices from the $\prod_{i=1}^m n_i - 1$ combinations, and T denotes the set of vertices in the last combination. We can generate a new vertex distribution, Θ, by moving an arbitrary vertex v from T to an arbitrary subset of S in Ψ. Then the attribute values of v are changed due to the movement. We denote these two combinations of attribute values of v before and after movement as η and ξ respectively.

If η and ξ contain the same values in some or all attributes, the situation is similar to that in Appendix 9.1. The set of graph vertices which have no common attribute values with v remains the same. Hence the number of zero elements in ΔP_A^2 is the same as that of Ψ. Therefore, in this situation, Θ is equivalent to Ψ, i.e., both give the lower bound of the number of zero elements in ΔP_A^2 as $(2n - \prod_{i=1}^m n_i) \times \prod_{i=1}^m (n_i - 1)$.

If η and ξ do not share any attribute values, we need to examine the three separate cases in the proof of Theorem 2.

Case 1 Due to the movement of v, the size of T is decreased by 1 and the size of S is increased by 1. The number of LB_1 in Eq. 14 is changed to

$$LB_1' = 2 \times \left(n - \prod_{i=1}^m n_i\right) \times \left(\prod_{i=1}^m (n_i - 1) + 1\right) \tag{18}$$

Case 2 Due to the movement of v, the size of T is decreased by 1 and the size of S_1 is increased by 1. The number of LB_2 in Eq. 15 is changed to

$$LB_2' = 2\left(\prod_{i=1}^m (n_i - 1) + 1\right) \times \left(\prod_{i=1}^m (n_i - 1) - 1\right) - \prod_{i=1}^m (n_i - 1) \times \prod_{i=1}^m (n_i - 2) \tag{19}$$

Case 3 Due to the movement of v, the size of S_2 remains unchanged. As a result, the number of LB_3 in Eq. 16 remains the same.

To sum up, the total number of zero elements in ΔP_A^2 after the weight adjustment is increased by

$$\Delta LB = 2 \times \left(n - 1 - \prod_{i=1}^{m} n_i \right) \tag{20}$$

Because $n \gg n_i$, $\Delta LB > 0$ when the vertex distribution changes from Ψ to Θ with v's movement. Therefore, we demonstrate that Ψ is the vertex distribution that gives the lower bound of the number of zero elements in ΔP_A^2. $\square$

References

Cai D, Shao Z, He X, Yan X, Han J (2005) Mining hidden community in heterogeneous social networks. In: Proceedings of Workshop on Link Discovery: Issues, Approaches and Applications (LinkKDD'05), pp 58–65, Chicago, IL

Cohn H, Kleinberg R, Szegedy B, Umans C (2005) Group-theoretic algorithms for matrix multiplication. In: Symposium on Foundations of Computer Science (FOCS)

Desikan P, Pathak N, Srivastava J, Kumar V (2005) Incremental page rank computation on evolving graphs. In: 14th International World Wide Web (WWW) Conference, pp 1094–1095

Hofmann T (1999) Probabilistic latent semantic indexing. In: Proceedings of SIGIR, pp 50–57

Jeh G, Widom J (2002) SimRank: a measure of structural-context similarity. In: Proceedings of KDD, pp 538–543

Long B, Zhang ZM, Wu X, Yu PS (2006) Spectral clustering for multi-type relational data. In: Proceedings of International Conference on Machine Learning (ICML), pp 585–592

Navlakha S, Rastogi R, Shrivastava N (2008) Graph summarization with bounded error. In: Proceedings of SIGMOD, pp 419–432

Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69:026113

Pons P, Latapy M (2006) Computing communities in large networks using random walks. J. Graph Algorithms Appl 10(2):191–218

Satuluri V, Parthasarathy S (2009) Scalable graph clustering using stochastic flows: applications to community discovery. In: Conference on Knowledge Discovery and Data Mining (KDD), pp 737–745

Shi J, Malik J (2000) Normalized cuts and image segmentation. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 22(8):888–905

Strassen V (1969) Gaussian elimination is not optimal. Numerische Mathematik 13:354–356

Sun J, Faloutsos C, Papadimitriou S, Yu PS (2007) Graphscope: parameter-free mining of large time-evolving graphs. In: Proceedings of KDD, pp 687–696

Sun Y, Han J, Zhao P, Yin Z, Cheng H, Wu T (2009) Rankclus: integrating clustering with ranking for heterogenous information network analysis. In: Proceedings of EDBT, pp 565–576

Tian Y, Hankins RA, Patel JM (2008) Efficient aggregation for graph summarization. In: Proceedings of SIGMOD, pp 567–580

Tong H, Faloutsos C, Pan J-Y (2006) Fast random walk with restart and its applications. In: Proceedings of ICDM, pp 613–622

Tong H, Faloutsos C, Pan J-Y (2008) Random walk with restart: fast solutions and applications. Knowl Inf Syst 14:327–346

Tsai C-Y, Chui C-C (2008) Developing a feature weight self-adjustment mechanism for a k-means clustering algorithm. Comput Stat Data Anal 52:4658–4672

Wang F, Li T, Wang X, Zhu S, Ding C (2011) Community discovery using nonnegative matrix factorization. Data Min Knowl Discov 22(3):493–521

Wu Y, Raschid L (2009) Approxrank: estimating rank for a subgraph. In: Proceedings of ICDE, pp 54–65

Xu X, Yuruk N, Feng Z (2007) Schweiger TAJ Scan: a structural clustering algorithm for networks. In: Proceedings of KDD, pp 824–833

Zhou Y, Cheng H, Yu JX (2009) Graph clustering based on structural/attribute similarities. In: Proceedings of the VLDB Endowment, pp 718–729

Zhou Y, Cheng H, Yu JX (2010) Clustering large attributed graphs: an efficient incremental approach. In: IEEE International Conference on Data Mining (ICDM), pp 689–698