Chapter 2
Clustering Analysis in Large Graphs with Rich Attributes

Yang Zhou and Ling Liu

DiSL, College of Computing, Georgia Institute of Technology,
Atlanta, Georgia, USA

Abstract. Social networks, communication networks, biological networks and many other information networks can be modeled as a large graph. Graph vertices represent entities and graph edges represent the relationships or interactions among entities. In many large graphs, there is usually one or more attributes associated with every graph vertex to describe its properties. The goal of graph clustering is to partition vertices in a large graph into subgraphs (clusters) based on a set of criteria, such as vertex similarity measures, adjacency-based measures, connectivity-based measures, density measures, or cut-based measures. Although graph clustering has been studied extensively, the problem of clustering analysis of large graphs with rich attributes remains a big challenge in practice. In this chapter we first give an overview of the set of issues and challenges for clustering analysis of large graphs with vertices of rich attributes. Based on the type of measures used for identifying clusters, existing graph clustering methods can be categorized into three classes: structure based clustering, attribute based clustering and structure-attribute based clustering. Structure based clustering mainly focuses on the topological structure of a graph for clustering, but largely ignore the vertex properties which are often heterogenous. Attribute based clustering, in contrast, focuses primarily on attribute-based vertex similarity, but suffers from isolated partitions of the graph as a result of graph clustering. Structure-attribute based clustering is a hybrid approach, which combines structural and attribute similarities through a unified distance measure. We argue that effective clustering analysis of a large graph with rich attributes requires the clustering methods to provide a systematic graph analysis framework that partition the graph based on both structural similarity and attribute similarity. One approach is to model rich attributes of vertices as auxiliary edges among vertices, resulting in a complex attribute augmented graph with multiple edges between some vertices. To show how to best combine structure and attribute similarity in a unified framework, the second part of this chapter will outline a cluster-convergence based iterative edge-weight assignment scheme that assigns different weights to different attributes based on how fast the clusters converge. We use a K-Medoids clustering algorithm to partition a graph into k clusters with both cohesive intra-cluster structures and homogeneous attribute values based on iterative weight updates. At each iteration, a series of matrix multiplication operations is used for calculating the random walk distances between graph

D.E. Holmes, L.C. Jain (Eds.): Data Mining: Found. & Intell. Paradigms, ISRL 23, pp. 7–27.
springerlink.com

vertices. Optimizations are used to reduce the cost of recalculating the random walk distances upon each iteration of the edge weight update. Finally, we discuss the set of open problems in graph clustering with rich attributes, including storage cost and efficiency, scalable analytics under memory constraints, distributed graph clustering and parallel processing.

1 Introduction

A number of scientific and technical endeavors are generating data that usually consists of a large number of interacting physical, conceptual, and societal components. Such examples include social networks, semantic networks, communication systems, the Internet, ecological networks, transportation networks, database schemas and ontologies, electrical power grids, sensor networks, research coauthor networks, biological networks, and so on. All the above networks share an important common feature: they can be modeled as graphs, i.e., individual objects interact with one another, forming large, interconnected, and sophisticated graphs with vertices of rich attributes. Multi-relational data mining finds the relational patterns in both the entity attributes and relations in the data. Graph mining, as one approach of multi-relational data mining, finds relational patterns in complex graph structures. Mining and analysis of these annotated and probabilistic graph structures is crucial for advancing the state of scientific research, accurate modeling and analysis of existing systems, and engineering of new systems.

Graph clustering is one of the most popular graph mining methodologies. Clustering is a useful and important unsupervised learning technique widely studied in literature [1,2,3,4]. The general goal of clustering is to group similar objects into one cluster while partitioning dissimilar objects into different clusters. Clustering has broad applications in the analysis of business and financial data, biological data, time series data, spatial data, trajectory data and so on. As one important approach of graph mining, graph clustering is an interesting and challenging research problem which has received much attention recently [5,6,7,8]. Clustering on a large graph aims to partition the graph into several densely connected components. This is very useful for understanding and visualizing large graphs. Typical applications of graph clustering include community detection in social networks, reduction of very large transportation networks, identification of functional related protein modules in large protein-protein interaction networks, *etc.* Although many graph clustering techniques have been proposed in literature, the problem of clustering analysis in large graphs with rich attributes remains to be challenging due to the demand on memory and computational resources and the demand on fast access to disk-based storage. Furthermore, with the grand vision of utility-driven and pay-as-you-go cloud computing paradigm shift, there is a growing demand for providing graph-clustering as a service. We witness the emerging interests from science and engineering fields in design and development of efficient and scalable graph analytics for managing and mining large information graphs.

Applications of graph clustering

In almost all information networks, graph clustering is used as a tool for analysis, modeling and prediction of the function, usage and evolution of the network, including business analysis, marketing, and anomaly detection. It is widely recognized by many that the task of graph clustering is highly application specific. In addition, by treating n-dimensional datasets as points in n-dimensional space, one can transform such n-dimensional datasets into graphs with rich attributes and apply graph theory to analyze the datasets. For example, modeling the World Wide Web (the Web) as a graph by representing each web page by a vertex and each hyperlink by an edge enables us to perform graph clustering analysis of hypertext documents and identify interesting artifacts about the Web, and visualize the usage and function of the Web. Furthermore, by representing each user as a vertex and placing (weighted) edges between two users as they communicate over the Internet services such as Skype, Microsoft's Messenger Live, and twitter, one can perform interesting usage statistics for optimizing related software and hardware configurations.

Concretely, in computer networks, clustering can be used to identify relevant substructures, analyze the connectivity for modeling or structural optimization, and perform root cause analysis of network faults [9,10]. In tele-communication systems, savings could be obtained by grouping a dense cluster of users on the same server as it would reduce the inter-server traffic. Similar analysis can help traditional tele-operators offer more attractive service packages or improve call delivery efficiency by identifying "frequent call clusters", i.e., groups of people that mainly call each other (such as families, coworkers, or groups of teenage friends) and hence better design and target the call service offers and special rates for calling to a limited set of pre-specified phone numbers. Clustering the caller information can also be used for fraud detection by identifying changes (outliers) in the communication pattern, call durations and a geographical embedding, in order to determine the cluster of "normal call destinations" for a specific client and which calls are "out of the ordinary". For networks with a dynamic topology, with frequent changes in the edge structure, local clustering methods prove useful, as the network nodes can make local decisions on how to modify the clustering to better reflect the current network topology [11]. Imposing a cluster structure on a dynamic network eases the routing task [12]. In bioinformatics, graph clustering analysis can be applied to the classification of gene expression data (e.g., gene-activation dependencies), protein interactions, and epidemic spreading of diseases (e.g., identifying groups of individuals "exposed" to the influence of a certain individual of interest or locating potentially infected people when an infected and contagious individual is encountered). In fact, cluster analysis of a social network also helps to identify the formation of trends or communities (relevant to market studies) and social influence behavior.

Graph Clustering: State of Art and Open Issues

Graph clustering has been studied by both theoreticians and practitioners over the last decade. Theoreticians are interested in investigating cluster properties,

algorithms and quality measures by exploiting underlying mathematical structures formalized in graph theory. Practitioners are investigating graph clustering algorithms by exploiting known characteristics of application-specific datasets. However, there is little effort on bridging the gap between theoretical aspect and practical aspect in graph clustering.

The goal of graph clustering is to partition vertices in a large graph into subgraphs (clusters) based on a set of criteria, such as vertex similarity measures, adjacency-based measures, connectivity-based measures, density measures, or cut-based measures. Based on the type of measures used for identifying clusters, existing graph clustering methods can be categorized into three classes: structure based clustering, attribute based clustering and structure-attribute based clustering. Structure based clustering mainly focuses on the topological structure of a graph for clustering, but largely ignores the rich attributes of vertices. Attribute based clustering, in contrast, focuses primarily on attribute-based vertex similarity, but suffers from isolated partitions of the graph as a result of graph clustering. Structure-attribute clustering is a hybrid approach, which combines structural similarity and attribute similarity through a unified distance measure. Most of the graph clustering techniques proposed to date aremainly focused on the topological structures using various criteria, including normalized cut [5], modularity [6], structural density [7] or flows [8]. The clustering results usually contain densely connected subgraphs within clusters. However, such methods largely ignore vertex attributes in the clustering process. On the other hand, attribute similarity based clustering [13] partitions large graphs by grouping nodes based on user-selected attributes and relationships. Vertices in one group share the same attribute values and relate to vertices in another group through the same type of relationship. This method achieves homogeneous attribute values within clusters, but ignores the intra-cluster topological structures. As shown in our experiments [14,15], the generated partitions tend to have very low connectivity.

Other recent studies on graph clustering include the following. Sun et al. [16] proposed GraphScope which is able to discover communities in large and dynamic graphs, as well as to detect the changing time of communities. Sun et al. [17] 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. [18] proposed a graph summarization method using the MDL principle. Tsai and Chiu [19] 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 simultaneously 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. [20] proposed an algorithm for mining communities on heterogeneous social networks. A method was designed for learning an optimal linear combination of different relations to meet users' expectation.

The rest of this chapter is organized as follows. Section 2 describes the basic concepts and general issues in graph clustering. Section 3 introduces the preliminary concepts and formulates the clustering problem for attribute augmented graphs and our proposed approach SA-Cluster. Section 4 presents presents our proposed incremental algorithm Inc-Cluster. Section 5 discusses optimization techniques to further improve computational performance. Finally, Section 6 concludes the chapter.

2 General Issues in Graph Clustering

Although graph clustering has been studied extensively, the problem of clustering analysis of large graphs with rich attributes remains to be a big challenge in practice. We argue that effective clustering analysis of a large graph with rich attributes requires a systematic graph clustering analysis framework that partition the graph based on both structural similarity and attribute similarity. One approach is to model rich attributes of vertices as auxiliary edges among vertices, resulting in a complex attribute augmented graph with multiple edges between some vertices.

In this section, we first describe the problem with an example. Then we review the graph clustering techniques and basic steps to take for preparation of clustering. We end this section by introducing the approach to combine structure and attribute similarity in a unified framework, called SA-Cluster. We dedicate Section 3 to present the design of SA-Cluster approach. The main idea is to use a cluster-convergence based iterative edge-weight assignment technique, which assigns different weights to different attributes based on how fast the clusters converge. We use a K-Medoids clustering algorithm to partition a graph into k clusters with both cohesive intra-cluster structures and homogeneous attribute values by applying a series of matrix multiplication operations for calculating the random walk distances between graph vertices. Optimization techniques are developed to reduce the cost of recalculating the random walk distances upon an iteration of the edge weight update.

The general methodology of graph clustering makes the following hypothesis [21]: First, a graph consists of dense subgraphs such that a dense subgraph contains more well-connected internal edges connecting the vertices in the subgraph than cutting edges connecting the vertices across subgraphs. Second, a random walk that visits a subgraph will likely stay in the subgraph until many of its vertices have been visited. Third, among all shortest paths between all pairs of vertices, links between different dense subgraphs are likely to be in many shortest paths. We will briefly review the graph clustering techniques developed based on each of the hypothesis.

The graph clustering framework consists of four components: modeling, measure, algorithm, and evaluation. The modeling component deals with the problem of transforming data into a graph or modeling the real application as a graph. The measurement deals with both distance measure and quality measure, both of which implement an objective function that determines and rates the quality

of a clustering. The algorithm is to exactly or approximately optimize the quality measure of the graph clustering. The evaluation component involves a set of metrics used to evaluate the performance of clustering by comparing with a "ground truth" clustering.

An *attribute-augmented graph* is denoted as $G = (V, E, \Lambda)$, where V is the set of n vertices, E is the set of edges, and $\Lambda = \{a_1, ..., a_m\}$ is the set of m attributes associated with vertices in V for describing vertex properties. Each vertex $v_i \in V$ is associated with an attribute vector $[a_1(v_i), ..., a_m(v_i)]$ where $a_j(v_i)$ is the attribute value of vertex v_i on attribute a_j, and is taken from the attribute domain $dom(a_j)$. We denote the set of attribute values by V_a and $V_a = \{a_j(v_i) \in dom(a_j) | i = 1, \ldots n, j = 1, \ldots m\}$. The **graph partition** of G is to partition an attribute-augmented graph G into k disjoint subgraphs, denoted by $G_i = (V_i, E_i, \Lambda)$, where $V = \bigcup_{i=1}^{k} V_i$ and $V_i \bigcap V_j = \phi$ for any $i \neq j$.

Figure 1 shows an example of a coauthor attributed Graph [15] where a vertex represents an author and an edge represents the coauthor relationship between two authors. In addition to the author ID, each vertex also has two attributes, research topic and age, associated with each author as the vertex properties. As shown in Figure 1, 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.

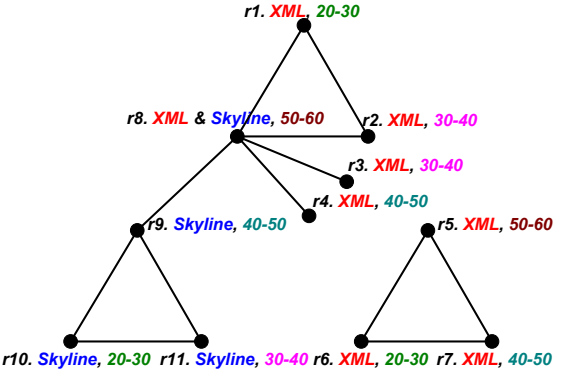


Fig. 1. A Coauthor Network with Two Attributes "Topic" and "Age" [15]

2.1 Graph Partition Techniques

Graph partition techniques refer to methods and algorithms that can partition a graph into densely connected subgraphs which are sparsely connected to each other. As we have discussed previously, there are three kinds of graph partition approaches: structure-similarity based graph clustering, attribute similarity based graph clustering, and structure-attribute combined graph clustering. Structure-based clustering only considers topological structure similarity but ignores the correlation of vertex attribute. Therefore, the clusters generated have a rather random distribution of vertex properties within clusters. On the

other hand, the attribute based clustering follows the grouping of compatible attributes and the clusters generated have good intra-cluster attribute similarity but a rather loose intra-cluster topological structure. A desired clustering of an attribute augmented graph should achieve a good balance between the following two properties: (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. The structure-attribute combined graph clustering method aims at partitioning a graph with rich attributes into k clusters with cohesive intra-cluster structures and homogeneous attribute values.

Orthogonal to the structure and attribute similarity based classification of graph clustering algorithms, another way to categorize graph clustering algorithms is in terms of top-down or bottom up partitioning. There are two major classes of algorithms: divisive and agglomerative. Divisive clustering follows top-down style and recursively splits a graph into subgraphs. In contrast, agglomerative clustering works bottom-up and iteratively merges singleton sets of vertices into subgraphs. The divisive and agglomerative algorithms are also called hierarchical since they produce multi-level clusterings, i.e., one clustering follows the other by refining (divisive) or coarsening (agglomerative). Most graph clustering algorithms proposed to date are divisive, including cut-based, spectral clustering, random walks, and shortest path.

The cut-based algorithms are associated with max-flow min-cut theorem [22], which states that "the value of the maximum flow is equal to the cost of the minimum cut". One of the earliest algorithms by Kernighan and Lin [23] splits the graph by performing recursive bisection (split into two parts at a time), aiming to minimize inter-cluster density (cut size). The high complexity of the algorithm ($O(|V|^3$) makes it less competitive in real applications. An optimization is proposed by Flake et al. [24] to optimize the bicriterion measure and the complexity, resulting in a more practical cut-based algorithm that is proportional to the number of clusters K using a heuristic.

The spectral clustering algorithms are based on spectral graph theory with Laplacian matrix as the mathematical tool. The proposition that the multiplicity k of the eigenvalue 0 of L equals to the number of connected components in the graph is used to establish the connection between clustering and spectrum of Laplacian matrix (L). The main reason for spectral clustering is that it does not make strong assumptions on the form of the clusters and can solve very general problems like intertwined spirals which k-means clustering handles poorly. Unfortunately, spectral clustering could be unstable under different choices of graphs and parameters [25,26]. The running complexity of spectral clustering equals to the complexity of computing the eigenvectors of Laplacian matrix which is ($O(|V|^3$).

The random walk based algorithms are based on the hypothesis that a random walk is likely to visit many vertices in a cluster before moving to the other cluster. The Markov clustering algorithm (MCL) by Van Dogen [21] is one of

the best in this category. The MCL algorithm iteratively applies two operators (expansion and inflation) by matrix computation until convergence. Expansion operator simulates spreading of random walks and inflation models demotion of inter-cluster walks; the sequence matrix computation results in eliminating inter-cluster interactions and leaving only intra-cluster components. The complexity of MCL is $O(m^2|V|)$, where m is the number of attributes associated to each vertex. A key point of random walk is that it is actually linked to spectral clustering [26], e.g., ncut can be expressed in terms of transition probabilities and optimizing ncut can be achieved by computing the stationary distribution of a random walk in the graph.

The shortest path based graph clustering algorithms are based on the hypothesis that the links between clusters are likely to be in the shortest paths. The use of betweenness and information centrality are two representative approaches in this category. The concept of edge betweenness [27] refers to the number of shortest paths connecting any pair of vertices that pass through the edge. Girvan and Newman [27] proposed an algorithm that iteratively removes one of the edges with the highest betweenness. The complexity of the algorithm is $O(|V||E|^2)$. Instead of betweenness, Fortunato et al. [28] used information centrality for each edge and stated that it performs better than betweenness but with a higher complexity of $O(|V||E|^3)$.

We firmly believe that no algorithm is a panacea for three reasons. First, the "best clustering" depends on applications, data characteristics, and granularity. Second, a clustering algorithm is usually developed to optimize some quality measure as its objective function, therefore, it is unfair to compare one algorithm that favors one measure with another that favors some different measure. Finally, there is no perfect measure that captures all the characteristics of cluster structures for all types of datasets. However, all graph clustering algorithms share some common open issues, such as storage cost, processing cost in terms of memory and computation, and the need for optimizations and distributed graph clustering algorithms for big graph analytics.

2.2 Basic Preparation for Graph Clustering

Graph Storage Structure. There are mainly three types of data structures for the representation of graphs in practice [29]: Adjacency list, Adjacency matrix, and Sparse Matrix. Adjacency list of a vertex keeps, for each vertex in the graph, a list of all other vertices to which it has an edge. Adjacency matrix of a graph G on n vertices is the $n \times n$ matrix where the non-diagonal entry a_{ij} is the number of edges from vertex i to vertex j, and the diagonal entry a_{ii}, depending on the convention, is either once or twice the number of edges (loops) from vertex i to itself. A sparse matrix is an adjacency matrix populated primarily with zeros. In this case, we create vectors to store the indices and values of the non-zero elements. The computational complexity of sparse matrix operation is proportional to the number of non-zero elements in the matrix. Sparse matrix is generally preferred because substantial memory requirement reductions can be realized by storing only the non-zero entries.

Handling A Large Number of Attributes. Large attributed graphs usually contain huge amounts of attributes in real applications. Each attribute may have abundant values. The available main-memory still remains very small compared to the size of large graphs with rich attributes. To make graph clustering approach applicable to a wide range of applications, we need to first handle rich attributes as well as continuous attributes with preprocessing techniques in the following.

First of all, we can perform correlation analysis to detect correlation between attributes and then perform dimensionality reduction to retain a smaller set of orthogonal dimensions. Widely used dimensionality reduction techniques such as principal component analysis (PCA) and multifactor dimensionality reduction (MDR) can be used to create a mapping from the original space to a new space with fewer dimensions. According to the mapping, we can compute the new attribute values of a vertex based on the values of its original attributes. Then we can construct the attribute augmented graph in the new feature space and perform graph clustering.

Discretization for Continuous Attributes. To handle continuous attributes, discretization can be applied to convert them to nominal features. Typically the continuous values are discretized into K partitions of an equal interval (equal width) or K partitions each with the same number of data points (equal frequency). For example, there is an attribute "prolific" for each author in the DBLP bibliographic graph indicating whether the author is prolific. If we use the number of publications to measure the prolific value of an author, then "prolific" is a continuous attribute. According to the distribution of the publication number in DBLP, we discretized the publication number into 3 partitions: authors with < 5 papers are labeled as low prolific, authors with ≥ 5 but < 20 papers are prolific, and the authors with ≥ 20 papers are tagged as highly prolific.

2.3 Graph Clustering with SA-Cluster

In order to demonstrate the advantage and feasibility of graph clustering with both structure similarity and attribute similarity, we describe the SA-Cluster proposed by Zhou et.al [14], a graph clustering algorithm by combining structural and attribute similarities. 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 self-adjustment method was used to learn the degree of contributions by different attributes in the graph clustering process based on clustering convergence rate. The attribute edge weights $\{\omega_1, \ldots, \omega_m\}$ are updated in each iteration of the clustering process. Accordingly, the transition probabilities on the graph are affected iteratively 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. Zhou et.al [14] showed through the experiments that the random walk distance computation takes 98% of the total clustering time in SA-Cluster.

The concept of random walk has been widely used to measure vertex distances and similarities. Jeh and Widom [30] designed a measure called SimRank, which defines the similarity between two vertices in a graph by their neighborhood similarity. Pons and Latapy [31] 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. [32] designed an algorithm for fast random walk computation. Other studies which use random walk with restarts include connection subgraph discovery [33] and center-piece subgraph discovery [34]. Liu et al. [35] proposed to use random walk with restart to discover subgraphs that exhibit significant changes in evolving networks.

In the subsequent sections, we describe in detail the SA-Cluster algorithm, especially the weight self-adjustment mechanism in [14] and the possible techniques for cost reduction through efficient computation of random walk distance upon the weight increments via incremental approaching the augmented graph[15]. We also provide a discussion on the set of open issues and research challenges for scaling large graph clustering with rich attributes.

3 Graph Clustering Based on Structural/Attribute Similarities

In this section, we first present the formulation of attribute augmented graph considering both structural and attribute similarities. A unified distance measure based on random walk is proposed to combine these two objectives. We then give an adaptive clustering algorithm SA-Cluster for the attributed graph.

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 properties, such as research topics, age, as well as other possible attributes like positions held and prolific numbers; 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.

To combine both structural and attribute similarities, we first define an *attributed augmented graph*. 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 to the attribute graph and form an attribute augmented graph. Authors with the topic ?XML? are connected to v_{11} in dashed lines. Similarly, authors with the topic ?Skyline? are connected to v_{12}. It intentionally omits the attribute vertices and edges corresponding to the age attribute, for the sake of clear presentation. Then the graph has two types of edges: the

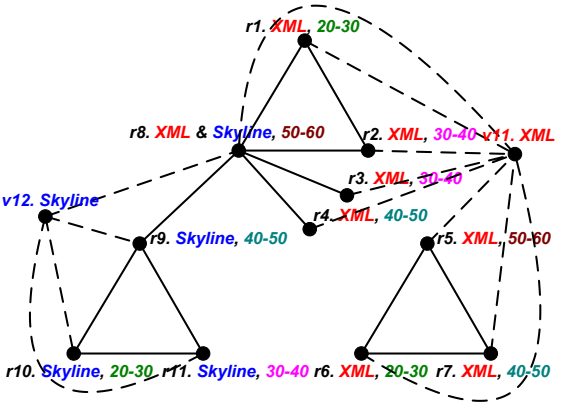


Fig. 2. Attribute Augmented Graph [15]

coauthor edge and the attribute edge. Two authors who have the same research topic are now connected through the attribute vertex.

A unified neighborhood random walk distance measure is designed to measure vertex closeness on an attribute augmented graph. The random walk distance between two vertices $v_i, v_j \in V$ is based on one or more paths consisting of both structure edges and attribute edges. Thus it effectively combines the structural proximity and attribute similarity of two vertices into one unified measure.

We first review the definition of transition probability matrix and random walk matrix. The transition probability matrix P_A is represented as

$$P_A = \begin{bmatrix} P_{V_1} & A_1 \\ B_1 & O \end{bmatrix} \tag{1}$$

where P_{V_1} is a $|V| \times |V|$ matrix representing the transition probabilities between structure vertices; A_1 is a $|V| \times |V_a|$ matrix representing the transition probabilities from structure vertices to attribute vertices; B_1 is a $|V_a| \times |V|$ matrix representing the transition probabilities from attribute vertices to structure vertices; and O is a $|V_a| \times |V_a|$ zero matrix.

The detailed definitions for these four submatrices are shown as follows. The transition probability from vertex v_i to vertex v_j through a structure edge is

$$p_{v_i,v_j} = \begin{cases} \dfrac{\omega_0}{|N(v_i)| * \omega_0 + \omega_1 + \ldots + \omega_m}, & if(v_i, v_j) \in E \\ 0, & otherwise \end{cases} \tag{2}$$

where $N(v_i)$ represents the set of structure vertices connected to v_i. Similarly, the transition probability from v_i to v_{jk} through an attribute edge is

$$p_{v_i,v_{jk}} = \begin{cases} \dfrac{\omega_j}{|N(v_i)| * \omega_0 + \omega_1 + \ldots + \omega_m}, & if(v_i, v_{jk}) \in E_a \\ 0, & otherwise \end{cases} \tag{3}$$

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

$$p_{v_{ik},v_j} = \begin{cases} \frac{1}{|N(v_{ik})|}, & if(v_{ik}, v_j) \in E_a \\ 0, & otherwise \end{cases} \tag{4}$$

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

Based on the definition of the transition probability matrix, the unified neighborhood random walk distance matrix R_A can be defined as follow,

$$R_A = \sum_{k=1}^{l} c(1-c)^k P_A^k \tag{5}$$

where P_A is the transition probability matrix of an attribute augmented graph G_a. l as the length that a random walk can go and $c \in (0, 1)$ is the random walk restart probability

According to this distance measure, we take a K-Medoids clustering approach to partition the graph into k clusters which have both cohesive intra-cluster structures and homogeneous attribute values. In the preparation phase, we initialize the weight value for each of the m attributes to value of 1, and select k initial centroids with the highest density values.

As different attributes may have different degrees of importance, at each iteration, a weight ω_i, which is initialized to 1.0, is assigned to an attribute a_i. A weight self-adjustment method is designed to learn the degree of contributions of different attributes. The attribute edge weights $\{\omega_1, \ldots, \omega_m\}$ are updated in each iteration of the clustering process through quantitatively estimation of the contributions of attribute similarity in the random walk distance measure. Theoretically we can prove that the weights are adjusted towards the direction of clustering convergence.

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 iteratively with the attribute weight adjustments. Thus the random walk distance matrix needs to be recalculated in next iteration of the clustering process. The algorithm repeats the above four steps until the objective function converges.

One issue with SA-Cluster is the computational complexity. We need to compute N^2 pairs of random walk distances between vertices in V through matrix multiplication. As $W = \{\omega_1, \ldots, \omega_n\}$ is updated, the random walk distances need to be recalculated, as shown in SA-Cluster. The cost analysis of SA-Cluster can be expressed as follows.

$$t \cdot (T_{random\_walk} + T_{centroid\_update} + T_{assignment}) \tag{6}$$

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

Algorithm 1. Attributed Graph Clustering SA-Cluster

Input: an attributed graph G, a length limit l of random walk paths, a restart probability c, a parameter σ of influence function, cluster number k.
Output: k clusters $V_1, ..., V_k$.

1: Initialize $\omega_1 = ... = \omega_m = 1.0$, fix $\omega_0 = 1.0$;
2: Calculate the unified random walk distance matrix R_A^l;
3: Select k initial centroids with highest density values;
4: Repeat until the objective function converges:
5: Assign each vertex v_i to a cluster C^* with a centroid c^* where $c^* = argmax_{c_j} d(v_i, c_j)$;
6: Update the cluster centroids with the most centrally located point in each cluster;
7: Update weights $\omega_1, ..., \omega_m$;
8: Re-calculate R_A^l;
9: Return k clusters $V_1, ..., V_k$.

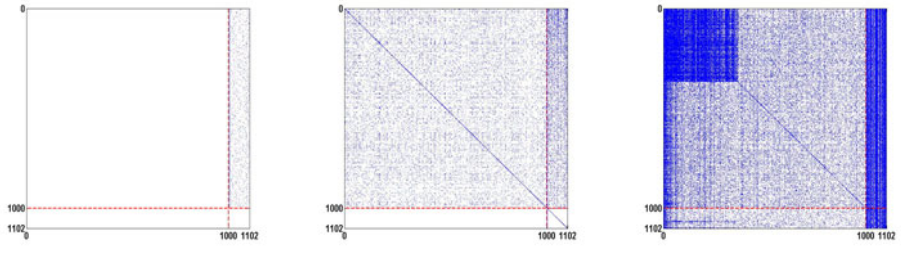
cost of updating cluster centroids, and $T_{assignment}$ is the cost of assigning all points to cluster centroids.

The time complexity of $T_{centroid\_update}$ and $T_{assignment}$ is $O(n)$, since each of these two operations performs a linear scan of the graph vertices. On the other hand, the time complexity of T_{random_walk} is $O(n^3)$ because the random walk distance calculation consists of a series of matrix multiplication and addition. According to the random walk equation, $R_A^l = \sum_{\gamma=1}^{l} c(1-c)^\gamma P_A^\gamma$ where l is the length limit of a random walk. To compute R_A^l, we have to compute P_A^2, P_A^3, ..., P_A^l, *i.e.*, $(l-1)$ matrix multiplication operations in total. It is clear that T_{random_walk} is the dominant factor in the clustering process. We find in the experiments that the random walk distance computation takes 98% of the total clustering time in SA-Cluster.

To reduce the number of matrix multiplication, full-rank approximation optimization techniques on matrix computation based on Matrix Neumann Series and SVD decomposition are designed to improve efficiency in calculating the random walk distance. It reduces the number of matrix multiplication from $O(l)$ to $O(log_2 l)$ where l is the length limit of the random walks.

4 The Incremental Algorithm

In this section, we show one way to improve the efficiency and scalability of SA-Cluster by using an efficient incremental computation algorithm to update the random walk distance matrix. 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 want to update the original random walk distance matrix, instead of re-calculating the matrix from scratch.



(a) ΔP_A^1 (b) ΔP_A^2 (c) ΔP_A^{20}

Fig. 3. Matrix Increment Series [15]

Example 1. Each of 1,000 authors has two attributes: "prolific" and "research topic". The first attribute "prolific" contains two values, 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 1st, 2nd, and 20th order matrix increment, 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. As shown in Figure 3, ΔP_A^k becomes denser when k increases, which demonstrates that the effect of attribute weight increments is propagated to the whole graph through matrix multiplication.

Existing fast random walk [32] or incremental PageRank computation approaches [36,37] can not be directly applied to our problem, as they partition the graph into a changed part and an unchanged part. However, our incremental computation problem is much more challenging than the above problems, 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. As we can see from Figure 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.

The main idea of the incremental algorithm [15] can be outlined as follows. According to Eq.(5), R_A is the weighted sum of a series of matrices P_A^k, where P_A^k is the k-th power of the transition probability matrix P_A, $k = 1, \ldots, l$. Hence the problem of computing ΔR_A can be decomposed into the subproblems of computing ΔP_A^k for different k values. Therefore, our target is, given the original matrix P_A^k and the edge weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$, compute the increment ΔP_A^k.

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

In summary, the incremental algorithm for calculating the new random walk distance matrix $R_{N,A}$, given the original R_A and the weight increments $\{\Delta\omega_1, \ldots, \Delta\omega_m\}$ iteratively computes the increments ΔP_A^k for $k = 1, \ldots, l$, and accumulates them into the increment matrix ΔR_A according to Eq.(5). 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

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

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

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

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

$$r \approx \frac{t \cdot T_{random\_walk}}{T_{random\_walk}} = t \tag{7}$$

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

Readers may refer to [14,15] for detailed experimental evaluation of the SA-Cluster and its incremental algorithm with structure-similarity based approach and attribute-similarity based approach in terms of runtime complexity and graph density and entropy measures.

5 Optimization Techniques

The iterative calculation of random walk distance matrix with attribute weight refinement is a very useful model for analyzing the closeness between two vertices in large graph data. Unfortunately, it has one significant drawback: memory performance is terrible. With large graph data, the results can be disastrous, leading to huge memory use, poor performance, and in the extreme case the termination of the clustering process by the operating system. The complexity of matrix multiplication, if carried out naively, is $O(n^3)$ so that calculating large numbers of matrix multiplications can be very time-consuming [29]. In this section, we will first analyze the storage cost of the incremental algorithm Inc-Cluster. Then we will discuss some techniques to further improve computational performance as well as save memory consumption.

5.1 The Storage Cost and Optimization

According to the incremental algorithm [15], we need to store a series of submatrices, as listed in the following.

The original transition probability matrix P_A. Based on the computational equations of ΔP_{V_k}, ΔA_k, ΔB_k and ΔC_k, we have to store P_{V_1}, B_1, ΔA_1 and $A_{N,1}$. According to the equation of ΔA_1, $\Delta A_1 = [\Delta\omega_1 \cdot A_{a_1}, ..., \Delta\omega_m \cdot A_{a_m}]$ where $A_1 = [A_{a_1}, A_{a_2}, ..., A_{a_m}]$. In addition $A_{N,1} = A_1 + \Delta A_1$. Therefore, we only need to store A_1 so as to derive ΔA_1 and $A_{N,1}$ with some simple computation. In summary, we need to store the original transition probability matrix P_A.

The (k-1)th order matrix increment ΔP_A^{k-1}. To calculate the kth order matrix increment ΔP_A^k, based on the equations of ΔP_{V_k}, ΔA_k, ΔB_k and ΔC_k, we need to use $\Delta P_{V_{k-1}}$, ΔA_{k-1}, ΔB_{k-1} and ΔC_{k-1}. Therefore, we need to store the (k-1)th order matrix increment ΔP_A^{k-1}. After ΔP_A^k is computed, we can throw away ΔP_A^{k-1} and save ΔP_A^k in turn for the computation of ΔP_A^{k+1} in the next iteration.

A series of A_k and C_k for $k = 2, ..., l$. In the equation of ΔA_k, we have derived $P_{V_{k-1}}\Delta A_1 = [\Delta\omega_1 \cdot A_{k,a_1}, ..., \Delta\omega_m \cdot A_{k,a_m}]$. We have mentioned that, the scalar multiplication $\Delta\omega_i \cdot A_{k,a_i}$ is cheaper than the matrix multiplication $P_{V_{k-1}}\Delta A_1$. In addition, this is more space efficient because we only need to store A_k, but not $P_{V_{k-1}}$. The advantage is that the size of A_k is $|V| \times |V_a| = n \times \sum_{i=1}^m n_i$, which is much smaller than the size of $P_{V_{k-1}}$ as $|V| \times |V| = n^2$. The above conclusion holds because $\sum_{i=1}^m n_i \ll n$. For example, in our experiments, we cluster a network of $10,000$ authors (i.e., $n = 10,000$) with 103 attribute vertices (i.e., $\sum_{i=1}^m n_i = 103$). The size of $P_{V_{k-1}}$ is $10,000^2$ while the size of A_k is $10,000 \times 103$. Thus $P_{V_{k-1}}$ is about 100 times larger than A_k.

Similarly, in the equation of ΔC_k, we have $B_{k-1}\Delta A_1 = [\Delta\omega_1 \cdot C_{k,a_1}, ..., \Delta\omega_m \cdot C_{k,a_m}]$. In this case we only need to store C_k, but not B_{k-1}. The advantage is that the size of C_k is $|V_a| \times |V_a| = (\sum_{i=1}^m n_i)^2$, which is much smaller than the size of B_{k-1} as $|V_a| \times |V| = \sum_{i=1}^m n_i \times n$. For example, in our experiments, the size of B_{k-1} is $103 \times 10,000$ while the size of C_k is 103^2. Thus B_{k-1} is about 100 times larger than C_k.

In summary, to calculate ΔP_A^k for $k = 2, ..., l$, we have to store A_k and C_k for different k values.

Total storage cost. By adding up the sizes of matrices P_A, ΔP_A^{k-1}, ΔP_A^k, R_A, A_k and C_k for different k, the total storage cost of Inc-Cluster is

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

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

$$T_{extra} = (l-1)(n \cdot \sum_{i=1}^{m} n_i + (\sum_{i=1}^{m} n_i)^2) \tag{8}$$

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

There are a number of research projects dedicated to graph databases. One objective of such development is to optimize the storage and access of large graphs with billions of vertices and millions of relationships. The Resource Description Framework (RDF) is a popular schema-free data model that is suitable for storing and representing graph data in a compact and efficient storage and access structure. Each RDF statement is in the form of subject-predicate-object expressions. A set of RDF statements intrinsically represents a labeled, directed graph. Therefore, widely used RDF storage techniques such as RDF-3X [40] and Bitmat [41] can be used to create a compact organization for large graphs.

5.2 Matrix Computation Optimization

There are three kinds of optimization techniques for matrix multiplication: block algorithms, sampling techniques and group-theoretic approach. All these techniques can speed up the performance of matrix computation.

Recent work by numerical analysts has shown that the most important computations for dense matrices are blockable [42]. The blocking optimization works well if the blocks fit in the small main memory. It is quite flexible and applicable to adjust block sizes and strategies in terms of the size of main memory as well as the characteristics of the input matrices.

Sampling Techniques are primarily meant to reduce the number of non-zero entries in the matrix and hence save memory. It either samples non-zero entries in terms of some probability distribution [43] or prunes those entries below the threshold based on the average values within a row or column [8]. When we use sparse matrix or other compression representations, it can dramatically improve the scalability and capability of matrix computation.

Recently, a fast matrix multiplication algorithm based on group-theoretic approach was proposed in [44]. It selects a finite group G satisfying the *triple product property* that allows $n \times n$ matrix multiplication to be reduced to multiplication of elements of the group algebra $\mathcal{C}[G]$. A Fourier transform is performed to decompose a large matrix multiplication into several smaller matrix multiplications, whose sizes are the character degrees of G. This gives rise to a complexity at least as great as $O(n^{2.41})$.

As mentioned previously, all experiments [15] on DBLP 84, 170 dataset needed a high-memory configuration (128GB main memory). To improve the applicability of clustering algorithms, Zhou et.al [45] showed their experimental results

about scaling large graph clustering with block multiplication optimization under memory constrained server. Experiments were done on a low-memory environment: two compute servers with 4GB and 8GB main memory, respectively. The runtime curve appears an interesting "U" curve. That is, the runtime is relatively long when the number of blocks is set to be extremely small or large but it keeps quite stable between the two ends of the spectrum. Although block optimization technique can dramatically decrease the memory consumption required, SA-Cluster with block multiplication optimization is usually 8 and 4.5 times slower than non-block SA-Cluster for 4GB and 8GB main memory environments respectively. One obvious approach to scaling large graph clustering is to develop distributed graph clustering methods. In short, we argue that how to improve the scalability and applicability of graph clustering algorithms continues to be an important open issue for wide deployment of graph clustering to big data analytics.

5.3 Parallelism

Although data storage techniques are growing at a much faster speed, the available main-memory still remains very small compared to the accessible disk-space. This creates the main bottleneck while executing large scale matrix computations. Parallel Computing has been employed for many years. Parallel approaches in a distributed memory multicomputer environment can efficiently improve the scalability and applicability of matrix multiplications [46]. The distributed system is viewed as a ring of processors and independent disk algorithms parallelized on block level.

MapReduce [47] is an excellent distributed computing model for processing large data sets on clusters of computers. For example, one can specify mappers that are responsible for distributing the block data to the reducers, with the help of a carefully chosen intermediate key structure, key comparator and partitioning functions. Reducers can be used to do the block multiplications and merge intermediate blocks to produce the final results.

6 Conclusion

In this chapter, we have given an overview of some of the essential techniques of graph clustering based through a classification of graph clustering methods into three classes: structure-based clustering, attribute-based clustering and structure-attribute combined clustering. We argue that a key challenge for addressing the problem of clustering large graphs with rich attributes is to achieve a good balance between structural and attribute similarities. Such balance can be driven by the objective function defined by domain-specific graph clustering analysis. We have described the use of the attribute-augmented graph to tackle graph clustering with rich attributes with two novel developments. First, we described a hybrid structure and attribute based neighborhood random walk distance measure, which is designed to measure vertex closeness on an attribute

augmented graph. Second, we describe the use of a learning algorithm to adjust the degree of contributions of different attributes in the random walk model as we iteratively refine the clusters, and prove that the weights are adjusted towards the direction of clustering convergence. By using a K-Medoids clustering approach, we show that the iterative weight assignment method is effective for partitioning a graph into k clusters with both cohesive intra-cluster structures and homogeneous attribute values. Finally, we presented a set of open issues and challenges in clustering analysis of large graphs with rich attributes, including processing and storage optimizations as well as distributed and parallel analytic models.

Acknowledgement. The authors are partially funded by research grants under NSF NetSE program, NSF CyberTrust program, an IBM SUR grant, an IBM faculty award, and a grant from Intel research council.

References

1. Ng, R., Han, J.: Efficient and effective clustering method for spatial data mining. In: Proc. 1994 Int. Conf. Very Large Data Bases (VLDB 1994), Santiago, Chile, pp. 144–155 (September 1994)
2. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD 1996), pp. 226–231 (1996)
3. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 1998), Seattle, WA, pp. 94–105 (June 1998)
4. Gibson, D., Kleinberg, J., Raghavan, P.: Inferring web communities from link topology. In: Proc. 9th ACM Conf. Hypertext and Hypermedia, Pittsburgh, PA, pp. 225–234 (June 1998)
5. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000)
6. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E69, 026113 (2004)
7. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: Scan: a structural clustering algorithm for networks. In: Proc. 2007 Int. Conf. Knowledge Discovery and Data Mining (KDD 2007), San Jose, CA, pp. 824–833 (August 2007)
8. Satuluri, V., Parthasarathy, S.: Scalable graph clustering using stochastic flows: Applications to community discovery. In: Proc. 2009 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD 2009), Paris, France (June 2009)
9. Wang, T., Srivatsa, M., Agrawal, D., Liu, L.: Learning indexing and diagnosing network faults. In: Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Paris, France, June 28-July 1, pp. 857–866 (2009)
10. Wang, T., Srivatsa, M., Agrawal, D., Liu, L.: Spatio-temporal patterns in network events. In: Proceedings of the 6th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2010), NY, USA, November 30-December 3 (2010)

11. Ramaswamy, L., Gedik, B., Liu, L.: A distributed approach to node clustering in decentralized peer-to-peer networks. IEEE Transactions on Parallel and Distributed Systems (TPDS) 16, 1–16 (2005)
12. Wang, Y., Liu, L., Pu, C., Zhang, G.: An utility-driven routing scheme for scalling multicast applications. In: Proceedings of 2009 International Conference on Collaborative Computing (CollaborateCom 2009), October 9-12. IEEE Press, Chicago (2009)
13. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proc. 2008 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 2008), Vancouver, Canada, pp. 567–580 (June 2008)
14. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. In: Proc. 2009 Int. Conf. on Very Large Data Base (VLDB 2009), Lyon, France (August 2009)
15. Zhou, Y., Cheng, H., Yu, J.X.: Clustering large attributed graphs: An efficient incremental approach. In: Proc. 2010 Int. Conf. on Data Mining (ICDM 2010), Sydney, Australia (December 2010)
16. Sun, J., Faloutsos, C., Papadimitriou, S., Yu, P.S.: Graphscope: parameter-free mining of large time-evolving graphs. In: Proc. 2007 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD 2007), San Jose, CA, pp. 687–696 (2007)
17. Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H., Wu, T.: Rankclus: Integrating clustering with ranking for heterogenous information network analysis. In: Proc. 2009 Int. Conf. Extending Database Technology (EDBT 2009), Saint Petersburg, Russia (March 2009)
18. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: Proc. 2008 ACM-SIGMOD Int. Conf. Management of Data (SIGMO 2008), Vancouver, Canada, pp. 419–432 (June 2008)
19. Tsai, C.-Y., Chui, C.-C.: Developing a feature weight self-adjustment mechanism for a k-means clustering algorithm. Computational Statistics and Data Analysis 52, 4658–4672 (2008)
20. Cai, D., Shao, Z., He, X., Yan, X., Han, J.: Mining hidden community in heterogeneous social networks. In: Proc. Workshop on Link Discovery: Issues, Approaches and Applications (LinkKDD 2005), Chicago, IL, pp. 58–65 (August 2005)
21. van Dongen, S.: Graph clustering by flow simulation. Ph.D. dissertation, University of Utrecht (2000)
22. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Canadian Journal of Mathematics 8, 399–404 (1956)
23. Kernighan, B.W., Lin, S.: An efficient heuristic procedur for partitioning graphs. Bell Syst. Techn. J. 49, 291–307 (1970)
24. Flake, G.W., Tarjan, R.E., Tsioutsiouliklis, K.: Graph clustering and minimum cut trees. Internet Mathematics 1 (2003)
25. Luxburg, U., Bousquet, O., Belkin, M.: Limits of spectral clustering. MIT Press, Cambridge (2005)
26. Luxburg, U.: A tutorial on spectral clustering. Technical Report 149 (2006)
27. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci., USA 99, 7821–7826 (2003)
28. Fortunato, S., Latora, V., Marchiori, M.: A method to find community structures based on information centrality. Phys. Rev. E.70, 056104 (2004)
29. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (2001)

30. Jeh, G., Widom, J.: SimRank: a measure of structural-context similarity. In: Proc. 2002 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2002), Edmonton, Canada, pp. 538–543 (July 2002)
31. Pons, P., Latapy, M.: Computing communities in large networks using random walks. Journal of Graph Algorithms and Applications 10(2), 191–218 (2006)
32. Tong, H., Faloutsos, C., Pan, J.-Y.: Fast random walk with restart and its applications. In: Proc. 2006 Int. Conf. on Data Mining (ICDM 2006), Hong Kong, pp. 613–622 (December 2006)
33. Faloutsos, C., McCurley, K., Tomkins, A.: Fast discovery of connection subgraphs. In: Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD 2004), Seattle, WA, pp. 118–127 (August 2004)
34. Tong, H., Faloutsos, C.: Center-piece subgraphs: problem definition and fast solutions. In: Proc. 2006 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD 2006), Philadelphia, PA, pp. 404–413 (2006)
35. Liu, Z., Yu, J.X., Ke, Y., Lin, X., Chen, L.: Spotting significant changing subgraphs in evolving graphs. In: Proc. 2008 Int. Conf. Data Mining (ICDM 2008), Pisa, Italy (December 2008)
36. Desikan, P., Pathak, N., Srivastava, J., Kumar, V.: Incremental page rank computation on evolving graphs. In: Proc. 2005 Int. World Wide Web Conf. (WWW 2005), Chiba, Japan, pp. 1094–1095 (May 2005)
37. Wu, Y., Raschid, L.: Approxrank: Estimating rank for a subgraph. In: Proc. 2009 Int. Conf. Data Engineering (ICDE 2009), Shanghai, China, pp. 54–65 (March 2009)
38. Strang, G.: Linear Algebra and its Applications. Brooks Cole (2005)
39. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
40. Neumann, T., Weikum, G.: Rdf-3x: a risc-style engine for rdf. In: Proceedings of the VLDB Endowment (PVLDB), vol. 1(1), pp. 647–659 (2008)
41. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "bit" loaded: a scalable lightweight join query processor for rdf data. In: Proceedings of the 19th International Conference on World Wide Web (WWW 2010), New York, NY, USA, pp. 41–50 (April 2010)
42. Press, W.H., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge (2007)
43. Cohen, E., Lewis, D.: Approximating matrix multiplication for pattern recognition tasks. In: Proceedings of the 8th Symposium on Discrete Algorithms (SODA 1997), New Orleans, Louisiana (Junuary 1997)
44. Cohn, H., Kleinberg, R., Szegedy, B., Umans, C.: Group-theoretic algorithms for matrix multiplication. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), Pittsburgh, Pennsylvania, USA (October 2005)
45. Zhou, Y., Liu, L.: Rapid: Resource-aware approach to large scale graph clustering. GT Tech Report, Atlanta, GA (April 2011)
46. Dackland, K., Elmroth, E.: Design and evaluation of parallel block algorithms: Lu factorization on an ibm 3090 vf/600j. In: Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Philadelphia, PA, USA (1992)
47. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, CA (December 2004)