Dense Subgraph Extraction with Application to Community Detection

Jie Chen and Yousef Saad

***Abstract*— This paper presents a method for identifying a set of dense subgraphs of a given sparse graph. Within the main applications of this "dense subgraph problem", the dense subgraphs are interpreted as communities, as in, e.g., social networks. The problem of identifying dense subgraphs helps analyze graph structures and complex networks and it is known to be challenging. It bears some similarities with the problem of reordering/blocking matrices in sparse matrix techniques. We exploit this link and adapt the idea of recognizing matrix column similarities, in order to compute a partial clustering of the vertices in a graph, where each cluster represents a dense subgraph. In contrast to existing subgraph extraction techniques which are based on a complete clustering of the graph nodes, the proposed algorithm takes into account the fact that not every participating node in the network needs to belong to a community. Another advantage is that the method does not require to specify the number of clusters; this number is usually not known in advance and is difficult to estimate. The computational process is very efficient, and the effectiveness of the proposed method is demonstrated in a few real-life examples.**

***Index Terms*— Dense subgraph, social network, community, matrix reordering, hierarchical clustering, partial clustering.**

I. Introduction

A challenging problem in the analysis of graph structures is the *dense subgraph problem*, where given a sparse graph, the objective is to identify a set of meaningful dense subgraphs. This problem has attracted much attention in recent years due to the increased interest in studying various complex networks, such as the World Wide Web (information network), social networks, and biological networks, etc. The dense subgraphs are often interpreted as "communities" [1]–[4], based on the basic assumption that a network system consists of a number of communities, among which the connections are much fewer than those inside the same community.

The recent data mining literature has seen various techniques for approaching this network analysis problem from different aspects. Because of a potentially wide variety of purposes, different definitions of communities are employed and methods are proposed, ranging from partitioning the network to minimize inter-connections between parts [5], [6], to aiming at extracting a large number of subgraphs that have a high enough density [7]–[9]. In addition to partitioning-based and density-based approaches, also seen are techniques that build hierarchical structures [10]–[13], that exploit stochastic block models [14]–[16], and that extract chains of adjacent cliques [17], [18]. It is beyond the scope of this paper to list the many existing approaches in such an emerging area. We refer the interested reader to surveys [19]–[21].

The authors are with the Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. Email: {jchen, saad}@cs.umn.edu

In this paper, we focus on the methodology of graph partitioning/clustering, with the goal of obtaining dense partitions/clusters. A broad set of partitioning techniques (spectral based [6], [22], multilevel based [23], [24], and stochastic based [25]) can be used. However, these methods have several drawbacks and issues that need to be addressed for the purpose of network analysis and community detection. The first drawback is that in general, the number k of partitions is a mandatory input parameter, and the partitioning result is sensitive to the change of k. In most applications, the number k is not known a priori. A number of researchers proposed to remedy this difficulty by tracking a goodness measure, such as the conductance [4] and the modularity [11], of the partitioning as a function of k. However, this remedy may not always be practical due to the underlying expensive computational cost of the algorithm. Second, most of these methods yield a *complete clustering* of the data. A crucial question when studying human behavior and social phenomena is: "Why should every participant be grouped into some community?" It is natural to consider that if a node in a network is far away from the rest of the nodes, then excluding this node from the analysis will usually yield more accurate results. Therefore, when attempting to discover communities, an *incomplete clustering* of the network is usually more desirable. Finally, note that many graph partitioning techniques based on optimizing an objective function [26] favor balancing, i.e., sizes of different partitions should not vary too much [5]. This may not accurately represent human communities, since it is common for social connections not to be evenly divided.

We propose a dense subgraph extraction approach that addresses the above issues. It is inspired by an effective technique designed for a similar problem—*matrix blocking* [27], [28]—from a different discipline (solving linear systems). How the proposed approach overcomes the general drawbacks of graph partitioning methodology for community detection will be made clear later in the paper. For now, let us consider the matrix blocking problem, which sheds light on the rationale of the approach we propose for the graph problem. For solving a linear system, preconditioning (e.g., an incomplete LU factorization) is an important step to improve the convergence of an iterative method [29], whereby blocking is a vital ingredient for preconditioning. The main motivation is that block preconditioning methods are known to yield better convergence than scalar ones (see [28], [29] and references therein). Matrix blocking amounts to symmetrically permuting the rows and the columns of a sparse matrix such that the nonzeros are moved toward the diagonal. In this way, the matrix exhibits dense diagonal blocks, whereas the rest of the area contains sporadic nonzeros (see Fig. 1(b)). In our case, each block naturally corresponds to a dense subgraph, or a community that we seek after.

The blocking algorithm presented in [28] groups similar columns (and rows) of a matrix according to a cosine similarity measure. By a non-trivial adaptation of this algorithm, we obtain what turns out to be a form of a hierarchical clustering (see, e.g. [30] for AGNES and DIANA) of the graph nodes using the same similarity measure. Hierarchical structures of a network have been exploited for the purpose of community extraction, by performing either a divisive clustering [11] or an agglomerative clustering [10], [31]. A feature of our method is that it can be viewed from both perspectives, by using the idea of a similarity graph G' computed from the original graph G. In the divisive perspective edges of G' are removed in an order of the edge weights, whereas in the agglomerative perspective edges are inserted to a set of isolated nodes in the opposite order to form G'. This approach avoids tracking/updating all-pairs distances in each merge or division step in the clustering process. The result is a computationally inexpensive procedure as long as the similarity scores can be efficiently computed.

Before looking at the algorithmic details, we shall mention here two important issues concerning this approach. The first concerns the similarity measure for building the hierarchy. Besides the obvious fact that the matrix blocking technique [28] (which inspires the algorithms proposed in this paper) uses the cosine similarity, this measure also has a clear interpretation for communities. A large cosine means that two nodes share a large portion of common neighbors with respect to their own neighbor sets, hence it can be interpreted as a probability that the two nodes belong to a community. This measure has been adopted in mining natural communities [31], and a similar measure—the Jaccard coefficient—is also used for a similar purpose [32]. The second is the interpretation of the hierarchy. Traditional hierarchical clustering methods cut the hierarchy at a specific level, yielding a complete partitioning of the graph. However, our goal is to identify *dense* subgraphs. Therefore, a sensible approach is to define the notion of the density, and to walk the hierarchy in a top-down fashion and return clusters only when they have high densities. By using this approach, one can navigate the hierarchy and adjust the density threshold (at almost no time cost) until a desirable result is achieved. Note that by introducing the definition of density, we do not intend to enumerate all the dense subgraphs; instead, we form an incomplete partitioning of the graph with dense partitions.

A common misperception is that computing pairwise similarities have at least a quadratic cost, which would make an algorithm based on such calculations ineffective for large data sets. Thus, to guarantee scalability, the shingling algorithm in [32] (which employs the Jaccard coefficient as the similarity measure) maps the set of neighbors of each graph node to a small number of "shingles", and the similarity of the nodes is translated to the number of shingles they share. In this paper, we employ a different approach. We exploit the fact that the matrix representation of the graph is sparse, and use sparse matrix computation techniques to compute the similarities in linear time. In fact, our overall algorithm is efficient. As will be seen in Sec. III, for a typical sparse graph, most parts of the algorithm are linear except that in addition we need to sort an array of size also linear in the number of graph nodes.

We note that existing community extraction approaches vary considerably in terms of applications and properties of the extracted subgraphs (e.g., ones that have the largest densities, the largest modularities, or bipartite structures, etc), which makes them difficult to compare, but in general a linear time algorithm equipped with external sorting, such as ours, will be desirable in facing mega- or giga-scale data. Further, as the study [21] points out, there tend to be a tradeoff between the quality of the subgraphs and the computational cost for existing methods. Thus, we show in Sec. IV extensive experiments demonstrating that our results are accurate and the extracted subgraphs are semantically meaningful.

II. Dense Subgraphs Extraction

Given a sparse graph $G(V,E)$ which consists of the vertex set V and the edge set E, we are interested in identifying dense subgraphs of G. To be precise, the candidate subgraphs should have densities higher than a threshold value in order to be interesting. We consider the following three types of graphs, with an appropriate definition of *density* for each one.

1) Undirected graphs. Undirected graphs are the most common models of networks, where the directions of the connections are unimportant, or can be safely ignored. A natural definition of the graph density is

$$d_G = \frac{|E|}{|V|(|V|-1)/2}. \tag{1}$$

Note that $d_G \in [0,1]$, and a subgraph has the density one if and only if it is a clique.

2) Directed graphs. The density of a directed graph is

$$d_G = \frac{|E|}{|V|(|V|-1)}, \tag{2}$$

since the maximum number of possible directed edges cannot exceed $|V|(|V|-1)$. In other words, the density of a directed graph also lies in the range from 0 to 1. It is interesting to note that if we "undirectify" the graph, i.e., remove the directions of the edges and combine the duplicated resulting edges, we yield an undirected graph $\tilde{G}(V,\tilde{E})$ with the edge set $\tilde{E}$. Then,

$$\frac{1}{2}d_{\tilde{G}} \le d_G \le d_{\tilde{G}}.$$

An immediate consequence is that if we extract the subgraphs of the undirected version of the graph given a density threshold, we essentially obtain directed subgraphs with densities at least half of the threshold.

3) Bipartite graphs. A bipartite graph is an undirected graph whose vertex set V can be partitioned in two disjoint subsets V_1 and V_2, such that every edge connects a vertex from V_1 and one from V_2. There are several reasons to consider bipartite graphs separately from general undirected graphs. The most important one is that a bipartite graph is generally used to model the connections between two different types of entities in a data set, such as the relationship between documents and terms, that between customers and products, etc. Also, as will soon be discussed, the proposed dense subgraph extraction algorithm for a general undirected graph does not directly apply to the bipartite graph case. Finally, the density of a bipartite graph, as computed by formula (1) can never reach one for bipartite graphs. Thus, we consider the following alternative definition for the density of a bipartite graph:

$$d_G = \frac{|E|}{|V_1|\cdot|V_2|}. \tag{3}$$

According to this definition, $d_G \in [0,1]$, and a subgraph has the density one if and only if it is a biclique.

The adjacency matrix A of the above three types of graphs has specific patterns. Throughout the paper, we assume that A is sparse, because we are considering subgraphs of a *sparse* graph. We also assume that the entries of A are either 0 or 1, since the weights of the edges are not taken into account for the density of a graph. In all cases, the diagonal of A is empty, since we do not allow self-loops. For undirected graphs, A is symmetric, whereas for directed graphs, A is only square. A natural matrix representation of a bipartite graph is a rectangular matrix B, where $B(i, j)$ is nonzero if and only if there is an edge connecting $i \in V_1$ and $j \in V_2$. The adjacency matrix for such a bipartite graph is indeed

$$A = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix},$$

where the vertices from V_1 are ordered before those from V_2. Note that there are situations where we do not know that the given undirected graph is bipartite in advance, i.e., A is given in a permuted form where the above 2×2 block structure is not revealed. In such a case, a simple strategy adapted from the breadth first search can be used to check if the inherent undirected graph is bipartite, and if so to extract the two disjoint subsets.

A. Matrix Blocking

As mentioned earlier, the dense subgraphs extraction methods proposed in this paper are inspired by the so-called matrix blocking problem. Fig. 1(b) illustrates a blocking result of a sparse matrix A. Here we describe a simple yet effective blocking algorithm [28] that accomplishes this result. It exploits the similarities between a pair of columns in the *pattern matrix* P of A. Recall that P is obtained from A by simply replacing its nonzero entries by ones. The idea is that the nonzero patterns of two columns corresponding to the same block should be more similar than those of the two columns that correspond to different blocks. To be specific, let some dense block of the reordered P correspond to a subset of vertices V_s. Also, let $i, j \in V_s$ and $k \notin V_s$; see Fig. 1(c). The heuristic is that the cosine of the angle between the i-th and the j-th columns of P is large, whereas that of the i-th and the k-th (or the j-th and the k-th) columns is small. The blocking algorithm is to find maximal subsets of V such that inside the same subset, for each vertex i, there exists a vertex $j \neq i$ such that the cosine of $P(:, i)$ and $P(:, j)$ is larger than a predefined threshold.

The adjacency matrix of an undirected graph plays exactly the same role as P here. Roughly speaking, the goal of dense subgraph extraction is to reorder the adjacency matrix and to find the dense diagonal blocks, each of which represents a dense subgraph. One is tempted to directly apply the above algorithm on the adjacency matrix of a given graph. However, a difficulty arises when choosing an appropriate similarity threshold. A further concern is that each block should employ a different threshold. For example, two columns corresponding to a larger block have a higher probability of yielding a larger cosine than those corresponding to a smaller block.

B. The Case of Undirected Graphs

Consider the matrix M that stores the cosines between any two columns of the adjacency matrix A:

$$M(i, j) = \frac{\langle A(:, i), A(:, j)\rangle}{\|A(:, i)\| \, \|A(:, j)\|}. \tag{4}$$

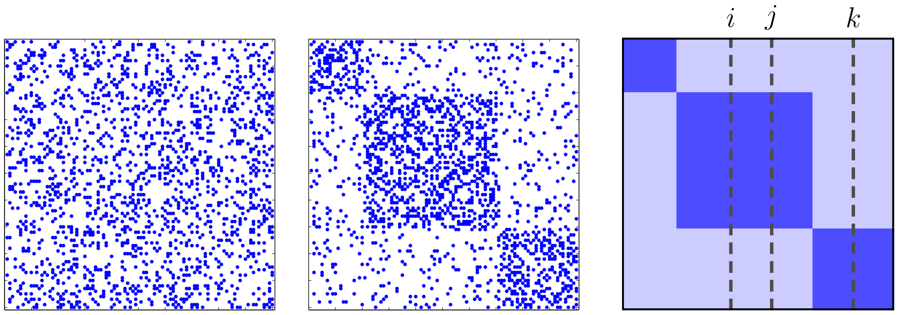
By reordering and partitioning the rows and columns of M in the same way as A, the above mentioned algorithm (by using a predefined similarity threshold) effectively yields a specially structured M: Entries outside the diagonal blocks of M, are all smaller than the threshold, whereas inside each non-trivial diagonal block, there exists at least one entry larger than the threshold for each row/column. Fig. 2(b) shows an illustration.

To avoid setting a fixed similarity threshold, we consider all possible ones as represented by the nonzero entries of M (excluding the diagonal). Going in ascending order of these entries, we set them to zero one by one. At some point after a few entries have been zeroed, M becomes a 2×2 block-diagonal matrix: the two off-diagonal blocks are completely zero (Fig. 2(c)). The last entry that was set to zero is a critical threshold, since by this value the rows and columns of M are partitioned in two subsets, and no smaller values can yield a partitioning. Once this initial partitioning is obtained, the zero-setting procedure is performed recursively on the two resulting partitions.

The above procedure can be precisely stated in the language of hierarchical divisive clustering. Given an undirected graph $G(V, E)$ and its adjacency matrix A, we construct a weighted graph $G'(V, E')$ whose weighted adjacency matrix M is defined in (4). Assume without loss of generality that G' is connected (otherwise process each connected component of G' separately). A top-down hierarchical clustering of the vertex set V is performed by successively deleting the edges $e' \in E'$, in ascending order of the edge weights. When G' first becomes disconnected, V is partitioned in two subsets, each of which corresponds to a connected component of G'. Then, the edge-removal process is continued on these two components to partition them in turn.

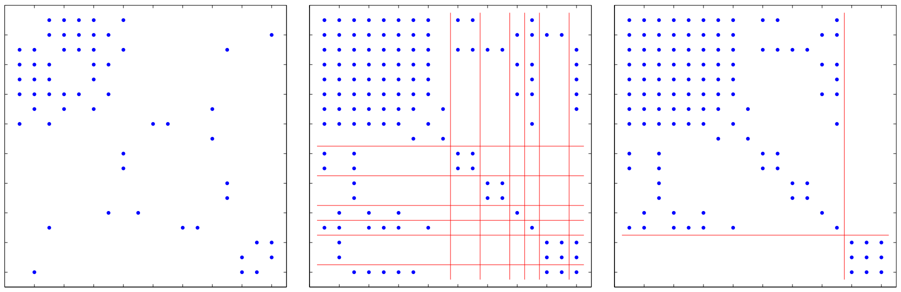
One detail that is left is to decide when to terminate the recursions. Recall that the objective is to find meaningful dense subgraphs of G. Therefore, termination will take place when the density of the partition passes a certain density threshold $d_{\min}$. Thus, a subset of the vertices is no longer partitioned if the corresponding subgraph has a density $\geq d_{\min}$. The only exception is that some subsets never meet this requirement and are recursively partitioned until they result in trivial subgraphs consisting of singletons. These singletons bear no interest and are ignored.

Algorithm 1 summarizes the proposed method. As an example, we consider a graph with 18 vertices and 29 edges as shown in Fig. 3(a). (This example comes from a figure in [4].) A visual inspection results that the graph has a dense component that contains vertices 1 to 7 (and possibly vertex 10), as well as a plausible dense, though small, component $\{14, 15, 16\}$. The first step is to compute the similarity matrix M and to sort its nonzero entries, as listed in (b). We construct the graph G' using the weighted adjacency matrix M. Starting from the smallest entry in the list, we remove edges of G' one by one until G' becomes disconnected. The two resulting subsets of vertices are: $\{1, \ldots, 13, 17, 18\}$ and $\{14, 15, 16\}$. The latter subset happens to yield a subgraph that has a density higher than the desired threshold 0.75. Hence, it is output as a dense subgraph. On the other hand, the former subset does not yield a subgraph satisfying the density requirement, so it is successively partitioned, until the subset $\{2, 5, 3, 1, 4, 7, 6\}$ is reached. This gives the other dense subgraph. Fig. 3(c) shows the resulting hierarchy/dendrogram. The output dense subgraphs are kept being partitioned in the hierarchy for illustration purposes.



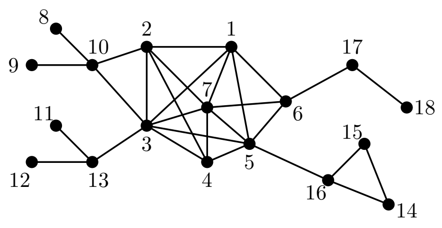
(a) A. (b) A after reordering. (c) Three columns of the reordered matrix A.

Fig. 1. A blocking of a sparse symmetric matrix A.



(a) A. (b) M and its blocking. (c) M with the first batch of nonzero entries set to zero.

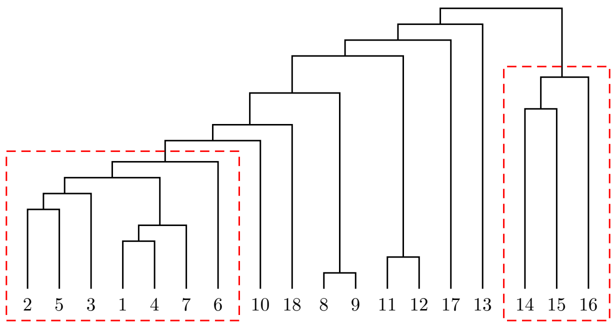
Fig. 2. An adjacency matrix A and its similarity matrix M. Plot (b) shows a partitioning of M by using a similarity threshold 0.5. Plot (c) shows the first partitioning of M in a recursive partitioning.



(a) The graph G.

i	j	$M(i,j)$
10	3	0.1890
10	5	0.2041
16	3	0.2182
⋮	⋮	⋮
4	1	0.8944
12	11	1.0000
9	8	1.0000

(b) Sorted nonzeros of M.



(c) The dendrogram.

Fig. 3. Two dense subgraphs (encapsulated in the red dashed frames) are found for a sparse undirected graph as shown in (a). The density threshold $d_{\min} = 0.75$.

C. The Case of Directed Graphs

The adjacency matrix A of a directed graph is square but not symmetric. When Algorithm 1 is applied to a non-symmetric adjacency matrix, it will result in two different dendrograms, depending on whether M is computed as the cosines of the columns of A, or the rows of A. There may be applications where the direction is required and one can choose to perform the analysis with either A (outgoing edges) or its transpose (incoming edges). However, in most applications, symmetrizing the graph is a sensible strategy because an incoming edge and outgoing edge have a similar "cost" (think in terms of communications in parallel algorithms for example). This is often performed for the somewhat related problem of graph partitioning for example. In the following we symmetrize the matrix A (i.e., replacing A by the pattern matrix of $A + A^T$) and use the resulting symmetric adjacency matrix to compute the similarity matrix M. The rest of the procedure follows Algorithm 1.

Note that this technique is equivalent to removing the directions of the edges in G, and extracting dense components from the undirected version of the graph. As discussed at the very beginning of this section, given an input parameter $d_{\min}$, the output directed subgraphs have densities guaranteed to be at least $d_{\min}/2$. If the occurrence of edge pairs (v_1, v_2) and (v_2, v_1), where v_1 and v_2 are two vertices, is rare, the densities of the output subgraphs will even be much higher.

Algorithm 1 Finding Dense Subgraphs of a Sparse Undirected Graph

```
Input: Sparse undirected graph $G$, density threshold $d_{\min}$.
 1: Construct $G'$ with the weighted adjacency matrix $M$ as
    defined in (4).
 2: Let $C$ be the array of tuples $(i, j, M(i,j))$, for all nonzero
    $M(i,j)$ and $i < j$, sorted in ascending order of $M(i,j)$.
 3: Run DENSE-SUBGRAPHS($G$, $G'$, $C$, $d_{\min}$).

 4: function DENSE-SUBGRAPHS($G$, $G'$, $C$, $d_{\min}$)
 5:     $k \leftarrow 0$
 6:     while $G'$ is connected do
 7:         Delete edge $\{C[k].i, C[k].j\}$.
 8:         $k \leftarrow k + 1$
 9:     end while
10:     Let the two connected components of $G'$ be
         $G'_s(V_s, E'_s)$ and $G'_t(V_t, E'_t)$.
11:     Let the two corresponding subgraphs of $G$ be
         $G_s(V_s, E_s)$ and $G_t(V_t, E_t)$.
12:     if $d_{G_s} \geq d_{\min}$ then
13:         Output $G_s$ as a dense subgraph.
14:     else if $|V_s| > 1$ then
15:         Let $C_s$ be the subarray of $C$, where $C_s[k].i \in V_s$
             and $C_s[k].j \in V_s$ for all $k$.
16:         DENSE-SUBGRAPHS($G_s$, $G'_s$, $C_s$, $d_{\min}$)
17:     end if
18:     Repeat lines 12–17 with $V_s$, $G_s$, $G'_s$, $C_s$ replaced
         by $V_t$, $G_t$, $G'_t$, $C_t$.
19: end function
```

D. The Case of Bipartite Graphs

Unfortunately, Algorithm 1 does not work for a bipartite graph where the vertex set V consists of two disjoint subsets V_1 and V_2. To see this, consider its adjacency matrix

$$A = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix},$$

where $B(i,j) = 1$ if there is an edge connecting $i \in V_1$ and $j \in V_2$, and $B(i,j) = 0$ otherwise. Then, the matrix M defined in (4) has the following form:

$$M = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}, \tag{5}$$

where M_1 (resp. M_2) contains the cosine similarities between the rows (resp. columns) of B. That is, without any edge removal of the graph G' (using M as the weighted adjacency matrix), the vertex set is already partitioned into two subsets: V_1 and V_2. Any subsequent hierarchical partitioning will only further subdivide these two subsets separately. This dilemma arises because we characterize the graph vertices inside a community by using the concept of "sharing neighbors". The only opportunity for two vertices to share common neighbors in a bipartite graph is that they both belong to a same subset V_i. However, when considering a subgraph which consists of vertices from a single V_i, this subgraph always has a zero density, thus eventually no subgraphs will be output from the algorithm.

One way to overcome the difficulty of Algorithm 1 when applied to bipartite graphs, is to perform a partial clustering separately for the rows and for the columns of B, by using the same similarity idea of Algorithm 1. In essence this is similar to the first approach suggested for directed graphs where the application warrants to differentiate between incoming and outgoing edges. It is equivalent to finding subsets of V_i where vertices share similar neighbors (from the complement of V_i). However, separate subsets do not directly imply a dense subgraph of the original bipartite graph. Alternatively, we opt to use an approach that shares the spirit of co-clustering: Find two subsets, $V_s \subset V_1$ and $V_t \subset V_2$, *simultaneously*, such that they are densely connected.

A reasonable strategy for this purpose is to augment the original bipartite graph by adding edges between some of the vertices that are connected by a path of length two. Clearly, this will add edges between vertices of the same V_i, making the graph a regular undirected graph. This will not change the density structure of the bipartite graph itself; rather, it encourages the discovery of the dense components. To see this, suppose V_s and V_t are densely connected. We can add enough edges between the vertices in V_s and also edges between those in V_t, then all the vertices in $V_s \cup V_t$ will appear so densely connected that $V_s \cup V_t$ can be easily extracted by a blocking algorithm. Fig. 4 illustrates an extreme case. The bipartite graph consists of three bicliques. If we artificially fill in edges between vertices inside the same biclique as shown in (b), then a blocking algorithm will easily recognize the three cliques in (c) and hence extract the three corresponding bicliques.

The question is what edges to add, since we do not know V_s and V_t. The similarity matrices M_1 (and M_2) in (5) are especially useful for answering this question. Consider two vertices, $v_{s_1}, v_{s_2} \in V_s$, for example. The fact that V_s and V_t are densely connected implies the high likelihood that v_{s_1} and v_{s_2} share similar neighbors. In other words, the two columns in A, which v_{s_1} and v_{s_2} correspond to, have a large cosine similarity. Therefore, it is natural to add an edge between v_{s_1} and v_{s_2}. From the perspective of the similarity matrix, this is to choose the largest entries of M and add them to A. To be precise, we modify the adjacency matrix A of a given bipartite graph into

$$\hat{A} = \begin{bmatrix} \hat{M}_1 & B \\ B^T & \hat{M}_2 \end{bmatrix}, \tag{6}$$

where $\hat{M}_1$ (resp. $\hat{M}_2$) is obtained by erasing the diagonal and keeping only the $2|E|$ largest nonzero entries of M_1 (resp. M_2), and $|E|$ is the number of edges in the original graph (i.e., it equals the number of nonzeros of B). Note that by keeping only the $2|E|$ largest nonzero entries, the number of edges in the augmented graph does not asymptotically increase.

Once the densification process yields the modified adjacency matrix $\hat{A}$, which represents the augmented graph, we proceed to calculate the similarity matrix $\hat{M}$:

$$\hat{M}(i,j) = \frac{\langle \hat{A}(:,i), \hat{A}(:,j) \rangle}{\|\hat{A}(:,i)\| \, \|\hat{A}(:,j)\|}, \tag{7}$$

which is used to build the hierarchy for the vertex set $V = V_1 \cup V_2$. Algorithm 2 summarizes the steps. Note that the procedure DENSE-SUBGRAPHS(G, G', C, $d_{\min}$) has been introduced in Algorithm 1.

A toy example is shown in Fig. 5. The blue-green coloring indicates the two disjoint subsets: $V_1 = \{1, 2, \dots, 8\}$ and $V_2 = \{9, 10 \dots, 13\}$. A visual inspection results that the bipartite graph consists of two dense components: all the vertices to the left

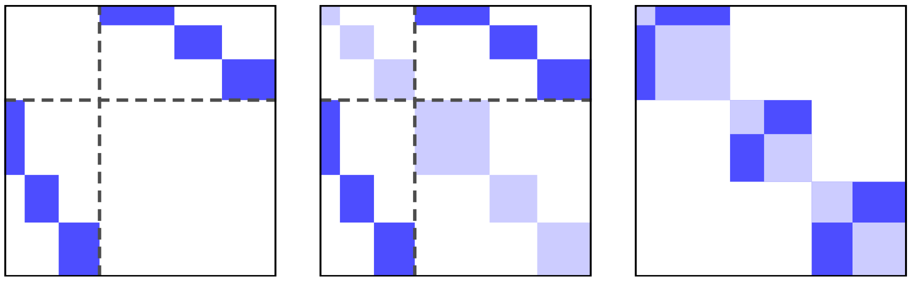

(a) Adjacency matrix A. (b) A after edge fill-in. (c) Blocking of the matrix in (b).

Fig. 4. A bipartite graph with the effect of edge fill-in.

Algorithm 2 Finding Dense Subgraphs of a Sparse Bipartite Graph

```
Input: Sparse bipartite graph $G$, density threshold $d_{\min}$.
1: [Densification:] Modify the adjacency matrix $A$ of $G$ into $\hat{A}$ as defined in (6).
2: Construct $G'$ with the weighted adjacency matrix $\hat{M}$ as defined in (7).
3: Let $C$ be the array of tuples $(i, j, \hat{M}(i,j))$, for all nonzero $\hat{M}(i,j)$ and $i < j$, sorted in ascending order of $\hat{M}(i,j)$.
4: Run DENSE-SUBGRAPHS($G$, $G'$, $C$, $d_{\min}$).
```

of 6 (including 6) contribute to one component, and the rest of the vertices (arranged in a hexagon shape) form the other. The densification of the graph, as shown in (b), further convinces the conjecture of the two dense components. Using the modified weighted adjacency matrix $\hat{A}$, we compute $\hat{M}$ and perform a hierarchical clustering similar to the one shown in Fig. 3(c). By using a density threshold $d_{\min} = 0.5$, it happens that two dense subgraphs are extracted, exactly the same as what we conjecture by visual inspection: $\{1, 2, 3, 9, 10, 11\}$ and $\{4, 5, 6, 7, 8, 12, 13\}$.

III. Implementation and Computational Costs

Despite the conceptual simplicity of the ideas described in the previous section, a careful design is needed to obtain efficient algorithms. This section discusses several important details that will transform the "conceptual" algorithms (Algo. 1 and 2) to more practical ones (Algo. 3 and 4). In particular, we address three issues: (A) How to efficiently compute the weighted adjacency matrix M ($\hat{M}$); (B) How to replace the costly routine DENSE-SUBGRAPHS by an equivalent but more efficient process for constructing the hierarchy; and (C) How to compute the densities for all the subgraphs in the hierarchy as needed for extracting dense ones. Each issue is discussed in a separate subsection, and we present the final improved algorithms at the end of this section.

The computational complexities of the proposed implementations will also be considered. As will soon be seen, most of the steps have a computational cost only linear to the number of vertices in the graph, except that in addition we need to sort an array of size also linear in this number. Thus, the proposed methods have the potential of being scalable to very large data sets. However, it is noted that there may be large prefactors in this simple big-O notation. As a result, to complement this incomplete theoretical analysis, we show in Section IV-B actual run times for a collection of real-life graphs from various application domains.

Some additional notation is needed. For a given graph $G(V, E)$, the number of vertices is $|V| = n$, and the number of edges is $|E|$. Since G is sparse, we typically assume that $|E| = O(|V|)$. We denote the number of nonzero entries of the adjacency matrix A, $\mathrm{nz}(A)$. For undirected graphs and bipartite graphs $\mathrm{nz}(A) = 2|E|$, and for directed graphs $\mathrm{nz}(A) = |E|$. In all cases, we have $\mathrm{nz}(A) = O(n)$.

A. The Computation of M ($\hat{M}$)

According to Eqn. (4), a naive way of computing M has the time complexity $O(\mathrm{nz}(A)^2) = O(n^2)$, since to compute an entry $M(i, j)$ takes time proportional to the sum of the numbers of nonzeros of $A(:, i)$ and $A(:, j)$. However, note that M is equal to $X^T X$, where X is the matrix A with each column normalized. A further investigation of Eqn. (6) (or (5)) indicates that the matrices M_1 and M_2 also take the form $X^T X$. Thus, an efficient way of computing M and $\hat{M}$ is to exploit the fact that X is sparse.

In the sequel, we consider how to multiply a sparse matrix X by its transpose:

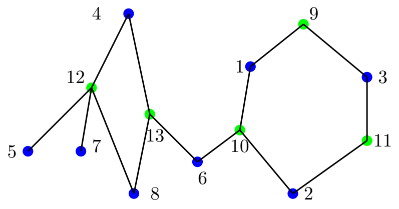
$$Y = X^T X := ZX,$$

where $Z = X^T$. The most efficient way in practice is to compute Y row by row. Note that

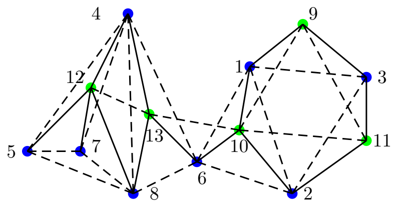
$$Y(i, :) = \sum_j Z(i, j) X(j, :).$$

Thus, we first transpose X into Z, then for each row i of Z, we compute a weighted sum of the rows of X which correspond to the nonzero elements in row i of Z. A particular issue is how to compute this weighted sum in time proportional to the total number of nonzeros involved, instead of to the length of a row of X. The technique is to pre-allocate two working arrays a and b, each of which has a size the same as a row of X. When computing row i of Y, we find the nonzero entries $Z(i, j)$, and for each j, we add the nonzeros of $X(j, :)$ multiplied by $Z(i, j)$ into the working array a, and store the information of which locations of a has been changed in the working array b. Then after the weighted sum is computed, we use the information in b to reset the array a to zero and also erase the content in b, then proceed to the next i.

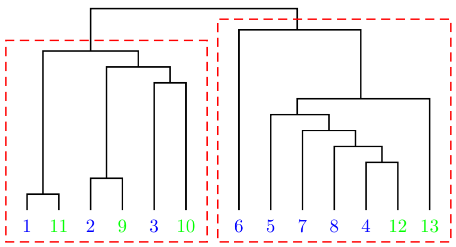
Let the maximum number of nonzeros per row of X be p. Then the upper bound of the time cost of the above technique for computing $X^T X$ is $O(p \cdot \mathrm{nz}(X))$, since to compute the i-th row of Y takes time $O(p \cdot \mathrm{nz}(Z(i, :)))$. In the average case, p can be considered a constant, thus the total time cost simplifies to $O(\mathrm{nz}(X))$. Also, transposing X has the same time complexity. In the graph language, X is the column-normalized A (or A^T), and p means the maximum number of neighbors for a vertex. Thus, the time cost of computing M (or $\hat{M}$) is $O(\mathrm{nz}(A)) = O(n)$.



(a) Bipartite graph G.



(b) Densification of G. Weights of the fill-in edges (dashed) have not been shown.



(c) The dendrogram constructed from (b).

Fig. 5. Two dense subgraphs (encapsulated in the red dashed frames) are found for a sparse bipartite graph as shown in (a). The density threshold $d_{\min} = 0.5$.

We should note that the above method is a standard technique used for sparse matrix-matrix multiplications (cf. e.g., [29]). Perhaps the only important point to stress here is the fact that *it takes time only linear in n to multiply two sparse matrices, assuming that the maximum number p of nonzeros per row is bounded by a constant*. Note that for some real-life graphs the degree of a vertex may follow a power low distribution, which means that p can become large for large graphs of a given application. It suffices to have one column/row pair of n entries for the cost of the product to rise to $O(n^2)$ (because the product becomes dense). Nevertheless, this situation is rare and it is also rare that p will be $O(n)$, and so the situation where the computational cost will rise to the forbidding $O(n^2)$ is rare in practice.

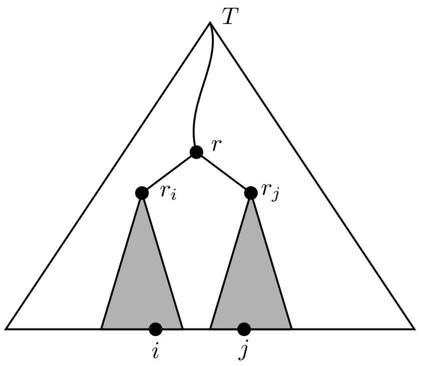


Fig. 6. The dendrogram T as a binary tree. Node r is the lowest common ancestor of i and j, and r_i and r_j are the children of r.

B. The Computation of the Hierarchy/Dendrogram

The routine DENSE-SUBGRAPHS (cf. Algorithm 1) essentially computes a hierarchy of the graph vertices in a top-down fashion. Recursive calls of this routine are very time consuming since between lines 5 and 9, with each removal of an edge in G', a graph traversal procedure (such as the breadth first search) is needed to examine the connectivity of the graph. However, as the two toy examples (cf. Fig. 3(c) and 5(c)) suggest, it is entirely possible to build the dendrogram T in an opposite (but equivalent) way: the bottom-up fashion.

The key is the array C which is sorted in ascending order of the nonzero entries $M(i,j)$ (or $\hat{M}(i,j)$)[1]. It indicates the order of the merges in the hierarchy/dendrogram T. Initially, each vertex $v \in V$ is a separate tree in the forest. Beginning from the end of the array C, each time we have a pair (i,j). We find the roots r_i and r_j of i and j, respectively. If r_i and r_j are different, we make a new root r with the left child r_i and the right child r_j (see Fig. 6). After iterating the whole array C, a single tree is returned, which is nothing but T.

Note that the above process is equivalent to monitoring the connected components of a graph when edges are successively inserted (a.k.a. incremental connected components [33]). Initially we have a virtual graph with the vertex set V but without edges. When reversely iterating the array C, we merge the two subsets s_i and s_j, which i and j belongs to respectively, if $s_i \neq s_j$. Finally, a single set, which contains all the vertices in V, is returned.

[1]To reduce the complication in reading, we thereafter omit the text "(or $\hat{M}$)" in this subsection. Readers are reminded that whenever the analysis is applied to a bipartite graph, all the notions involving M should be replaced by $\hat{M}$.

Therefore, we can utilize the two standard disjoint-set operations SET-FIND and SET-UNION to assist the process of building T. When we iterate C and get a pair (i,j) each time, we first do SET-FIND$(i) \to s_i$ and SET-FIND$(j) \to s_j$. If $s_i = s_j$, nothing is done. On the other hand, if $s_i \neq s_j$, we call SET-UNION to combine s_i and s_j. Meanwhile, we make a new node r which has the r_i and r_j (stored with the disjoint-set data structure) as the two children, and associate r with the combined set $s = s_i \cup s_j$.

The total time of the above process can be split in two parts: (a) the time of all the SET-FIND and SET-UNION calls, and (b) the gross time to build T. Part (a) is indeed the incremental connected component process, which takes time $O(n + \mathrm{nz}(M))$, since the graph G' has n vertices and $O(\mathrm{nz}(M))$ edges. Part (b), which consists of making new nodes and assigning children, has a time complexity linear to the size of T, which is $O(n)$.

We still can improve the performance. Recall that the whole bottom-up process is nothing but to yield the graph G' from a collection of isolated vertices by successively inserting edges. We can stop the insertion of edges at some point. This essentially yield an incomplete hierarchy, which is the part of T below some level. We opt to stop after we have inserted $O(n)$ edges. In practice, the number of inserted edges can be simply set as $\mathrm{nz}(A)$, or as $\tau \cdot \mathrm{nz}(A)$ by introducing some coefficient parameter τ. This may greatly reduce the cost of part (a) from $O(n + \mathrm{nz}(M))$ to $O(n)$, and also some minimal cost of part (b). By doing this, the negative impact on the final dense subgraphs extraction process is hoped to be minimal, since we only miss, if any, large subgraphs that have not been formed by merging in the hierarchy. We still are able to extract the dense parts of the hypothetically missing large subgraphs. Another advantage is that instead of sorting the nonzeros of M in $O(\mathrm{nz}(M)\log(\mathrm{nz}(M)))$ time to make the array C, we only need to sort the $O(n)$ largest nonzeros in $O(n\log n)$ time (plus finding the largest nonzeros in $O(\mathrm{nz}(M))$ time, which is negligible compared with $O(n\log n)$.)

C. Collecting Density Information and Extracting Subgraphs

Recall that in the hierarchy T, each internal node r represents a subgraph of G whose vertices are the leaf nodes of the subtree rooted at r. The dense subgraphs extraction process starts from visiting the root of T. If the subgraph corresponding to the current node has the density higher than the input threshold $d_{\min}$, it is output; otherwise the two children of the current node are visited and the whole process is recursive. Thus, the extraction process is equivalent to a traversal of T and is very cheap, given that the densities of all the subgraphs have been computed and stored in the internal nodes r.

In the following we discuss how the subgraph densities are computed. For each internal node r of T, it is sufficient to store two values: the number n_r of vertices the corresponding subgraph contains, and the number e_r of edges. The number of vertices can be easily computed in a recursive way: $n_r = n_{r_i} + n_{r_j}$, where r_i and r_j are the two children of r. However, the computation of e_r is not that straightforward. It is the sum of e_{r_i}, e_{r_j} and $e_{c(r_i,r_j)}$, where $e_{c(r_i,r_j)}$ is the number of edges crossing the subgraphs r_i and r_j represent. Thus, the computation of e_r can be split in two phases. The first phase is to compute $e_{c(r_i,r_j)}$ for each internal node r. The second phase is to recursively compute $e_r = e_{r_i} + e_{r_j} + e_{c(r_i,r_j)}$ for node r from its two children.

Further explanations on how $e_{c(r_i,r_j)}$ is counted are in order. Recall in Fig. 6 that r is the lowest common ancestor (LCA) of i and j. Thus, we initialize $e_r = 0$ for all r. For each edge $\{i,j\}$ in the graph, we find the lowest common ancestor r of i and j and add 1 to e_r. After iterating all the edges, the temporary e_r value for each internal node r in the hierarchy is exactly $e_{c(r_i,r_j)}$, thus finishing phase one as mentioned in the previous paragraph.

Currently, the most efficient LCA data structure answers queries in constant time after a linear time preprocessing [34], [35]. Thus, the time cost for phase one is $O(n+\mathrm{nz}(A)) = O(n)$, since the tree T has $O(n)$ nodes and we need to find the lowest common ancestors for $O(\mathrm{nz}(A))$ pairs. This complexity applies to all the three types of graphs, since even after modifications, the adjacency matrix of the graph always has $O(\mathrm{nz}(A))$ nonzeros. Therefore, the time cost of computing n_r and e_r for all nodes r in the hierarchy takes time $O(n)$. This is also the cost of the final dense subgraphs extraction process, which simply consists of a traversal of T.

D. The Final Algorithms

In summary, the improved versions of the two algorithms presented in Sec. II are shown in Algorithms 3 and 4, by incorporating the above discussions. These supersede Algorithms 1 and 2 in the rest of the paper. In the pseudocodes, the hierarchy/dendrogram T is a tree with the root $T.root$. A node r in the tree has the left child $left$, the right child $right$, and the density $density$ which is computed from num_vertex (n_r) and num_edge (e_r) according to the appropriate definition of density introduced at the beginning of Sec. II.

Algorithm 3 Finding Dense Subgraphs of a Sparse Undirected Graph (equivalent to Algorithm 1, more efficient)

```
Input: Sparse undirected graph $G$, density threshold $d_{\min}$.
1: Compute the matrix $M$ as defined in (4).
2: Sort the largest $t$ nonzero entries of $M$ in ascending order,
   where $t = \mathrm{nz}(A)$. Denote $C$ the sorted array.
3: Construct the hierarchy $T$ according to the sorted vertex pairs
   designated by $C$.
4: COUNT-VERTICES-AND-EDGES($T$, $G$)
5: Compute $r.density$ for all nodes $r$ of $T$ according to (1).
6: EXTRACT-SUBGRAPHS($T.root$)

7: function COUNT-VERTICES-AND-EDGES($T$, $G$)
8:     Initialize $r.num\_edge \leftarrow 0$ for all nodes $r$ of $T$.
9:     Construct the LCA data structure for $T$.
10:    for all edge $\{i,j\}$ of $G$ do
11:        Find the lowest common ancestor $r$ of $i$ and $j$.
12:        $r.num\_edge \leftarrow r.num\_edge + 1$
13:    end for
14:    COUNT-VERTICES-AND-EDGES-WRAP-UP($T.root$)
15: end function

16: function COUNT-VERTICES-AND-EDGES-WRAP-UP($r$)
17:    if $r.left \neq nil$ and $r.right \neq nil$ then
18:        COUNT-VERTICES-AND-EDGES-WRAP-UP($r.left$)
19:        COUNT-VERTICES-AND-EDGES-WRAP-UP($r.right$)
20:    end if
21:    if $r.left \neq nil$ and $r.right \neq nil$ then
22:        $r.num\_vertex \leftarrow$
             $r.left.num\_vertex + r.right.num\_vertex$
23:        $r.num\_edge \leftarrow$
             $r.left.num\_edge + r.right.num\_edge + r.num\_edge$
24:    else
25:        $r.num\_vertex \leftarrow 1$
26:    end if
27: end function

28: function EXTRACT-SUBGRAPHS($r$)
29:    if $r.density > d_{\min}$ then
30:        Output the leaves of the subtree rooted at $r$.
31:    else if $r.left \neq nil$ and $r.right \neq nil$ then
32:        EXTRACT-SUBGRAPHS($r.left$)
33:        EXTRACT-SUBGRAPHS($r.right$)
34:    end if
35: end function
```

IV. Experimental Results and Applications

This section shows extensive experimental results to illustrate the efficiency and the effectiveness of the proposed algorithms for extracting dense subgraphs. The experiments were performed under a Linux desktop with four AMD Opteron Processors (2.20GHz) and 16GB memory. The programs were not parallel and used only one processor. The algorithms were implemented in C/C++, and the programs were compiled using g++ with -O2 level optimization.

A. Simulations and Accuracies

In this subsection we show the dense subgraphs extraction results of two simulated graphs. A visualization is shown in

Algorithm 4 Finding Dense Subgraphs of a Sparse Bipartite Graph (equivalent to Algorithm 2, more efficient)

```
Input: Sparse bipartite graph $G$, density threshold $d_{\min}$.
1: [Densification:] Modify the adjacency matrix $A$ of $G$ into $\hat{A}$ as defined in (6).
2: Compute the matrix $\hat{M}$ as defined in (7).
3: Sort the largest $t$ nonzero entries of $\hat{M}$ in ascending order, where $t = \mathrm{nz}(\hat{A})$. Denote $C$ the sorted array.
4: Construct the hierarchy $T$ according to the sorted vertex pairs designated by $C$.
5: COUNT-VERTICES-AND-EDGES($T$, $G$). [Instead of counting the number of vertices $r.num\_vertex$ for each subgraph, count the number of vertices that belong to each partite set for each subgraph, in a similar way.]
6: Compute $r.density$ for all nodes $r$ of $T$ according to (3).
7: EXTRACT-SUBGRAPHS($T.root$)
```

Fig. 7. The graphs were randomly generated subject to the parameters given in Tab. I. The simulated undirected graph has three dense components/subgraphs, and the bipartite graph has four. We computed the densities of the dense components for each graph, and used the smallest of the densities as the input parameter $d_{\min}$ to our algorithms. The aim of this experiment is to show that the proposed algorithms are able to discover the intended dense components when a good parameter is provided. Other experiments for the situation when the density threshold is unknown in advance will be discussed in later subsections.

TABLE I

SIMULATION PARAMETERS FOR THE GRAPHS IN FIG. 7. FOR THE UNDIRECTED GRAPH, EACH (s, t) PAIR MEANS A (SUB)GRAPH WITH s VERTICES AND APPROXIMATELY t EDGES. FOR THE BIPARTITE GRAPH, EACH (s_1, s_2, t) PAIR MEANS A (SUB)GRAPH WITH $s_1 + s_2$ VERTICES AND APPROXIMATELY t EDGES.

Graph	Undirected	Bipartite
Whole	(100, 2000)	(100, 170, 1940)
Component 1	(25, 420)	(20, 40, 370)
Component 2	(30, 550)	(20, 35, 280)
Component 3	(20, 290)	(17, 30, 260)
Component 4		(15, 45, 340)

The criterion we use to measure the "accuracy" of the extracted dense subgraphs is the F-score. Here, the term "accuracy" only states how much the extracted subgraphs deviate from the intended dense components. Indeed, a precise determination of the dense subgraphs in each simulated case does not exist. As long as the output subgraphs have densities higher than the input threshold, there is no harm in considering that the result is as good as the "ground truth". For each dense component i in the intended construction, let V_i be its vertex set. We compare V_i with the extraction result $\tilde{V}_i$, and the F-score is defined as

$$F_i = \frac{2}{\dfrac{1}{precision} + \dfrac{1}{recall}} = \frac{2}{\dfrac{|V_i|}{|V_i \cap \tilde{V}_i|} + \dfrac{|\tilde{V}_i|}{|V_i \cap \tilde{V}_i|}}.$$

Tab. II shows the average F-score for each component i by simulating the graphs 100 times. It can be seen that the extraction results match the intended constructions quite well.

TABLE II

ACCURACY OF THE EXTRACTED DENSE SUBGRAPHS. THE UPPER TABLE IS FOR THE UNDIRECTED GRAPH, AND THE BOTTOM ONE IS FOR THE BIPARTITE GRAPH.

Dense component	1	2	3
Average F-score	0.9844	0.9882	0.9694

Dense component	1	2	3	4
Average F-score	0.9720	0.9310	0.9755	0.9730

B. Real Graphs and Running Times

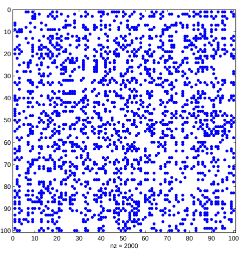
We tested the performance of our algorithms on real-life graphs with different sizes and from various application domains. The graphs are listed in Tab. III; they include a social network (`polblogs`), a biological network (`yeast`), a citation network (`hep`), a trust network (`epinions`), an information network (`NDwww`), and graphs that represent the relationships between words (`Reuters911`, `foldoc`, `dictionary28`), between users and movies (`MovieLens`), and between words and documents (`newsgroup`, `cmuSame`, `cmuDiff`, `cmuSim`). In this subsection, we are mainly interested in the running times of the algorithms as opposed to the graph sizes. Some of the graphs will be mentioned again in later subsections for analyzing the extraction results and understanding community structures. For such graphs, more information related to the semantics of the graphs will be presented when appropriate.

TABLE III

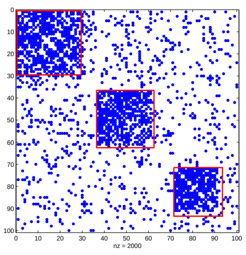
SOME REAL-LIFE GRAPHS.

Graph	Description
`polblogs` [36]	A directed network of hyperlinks between weblogs on US politics.
`yeast` [37]	Protein-protein interaction network.
`Reuters911` [38]	Reuters terror news network.
`foldoc`	Free on-line dictionary of computing. `http://www.cise.ufl.edu/research/sparse/matrices/Pajek/foldoc.html`
`hep`	The citation graph of the hep-th portion of arXiv. `http://www.cs.cornell.edu/projects/kddcup/datasets.html`
`epinions` [39]	Trust network of the users on Epinions.com.
`dictionary28`	Dictionary. `http://www.cise.ufl.edu/research/sparse/matrices/Pajek/dictionary28.html`
`NDwww` [40]	Webpages within nd.edu domain.
`cmuSame` [41] `cmuDiff`, `cmuSim`	The 20 Newsgroups data set (three subsets).
`MovieLens` [42]	The MovieLens data set.
`newsgroup` [43]	The 20 Newsgroups data set.

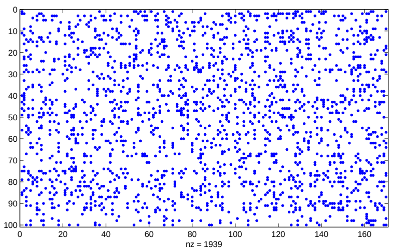
The running times are shown in Tab. IV. Two aspects of the experimental design are noted. First, the density threshold $d_{\min}$ is the least important parameter in this experiment, since it affects only the extraction time (the last column in the table), which is almost negligible compared with other times. This meanwhile indicates that the parameter $d_{\min}$ does not constitute a weakness of our algorithms—we can always tune the parameter in real time. We fixed $d_{\min}$ to be 0.1 in this experiment. The second aspect is the parameter τ, where recall that in Sec. III-B we insert $\tau \cdot \mathrm{nz}(A)$ edges in the incremental connected component process. This constructs an incomplete, yet probably sufficient, hierarchy T. The parameter τ directly affects the sorting time and the time



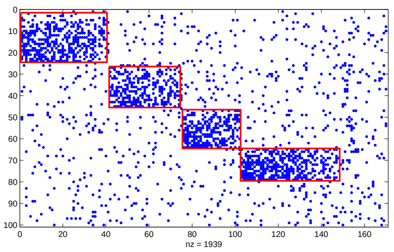
(a) An undirected graph.



(b) Dense subgraphs of (a).



(c) A bipartite graph.

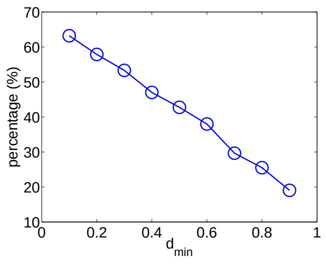


(d) Dense subgraphs of (c).

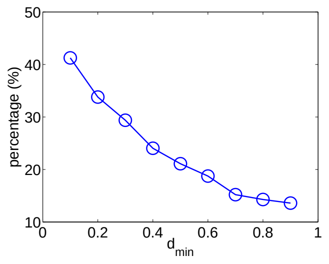
Fig. 7. The extracted dense subgraphs of two simulated graphs.

to compute the hierarchy. In most of the cases $\tau = 1$ is sufficient to yield meaningful dense subgraphs, except that in a few cases we tune the parameter to an appropriate value such that desirable subgraphs are extracted. The values of τ are listed in the table.

From Tab. IV we see that the proposed algorithms are efficient. A large part of the running time is spent on the matrix-matrix multiplication (computing M or $\hat{M}$), which is not difficult to parallelize. Note that all the graphs are run on a single desktop machine. In the future we will investigate parallel versions of the algorithms that can deal with massive graphs.



(a) `hep`



(b) `dictionary28`

Fig. 9. Percentage of vertices that belong to the extracted dense subgraphs.

C. *Power Law Distribution of the Dense Subgraph Sizes*

To further understand the extraction results, we plot in Fig. 8 the distribution of the dense subgraph sizes. We experimented with two graphs: a collaboration network (`hep`) and a dictionary graph (`dictionary28`), using various density thresholds. Within each plot, the horizontal axis is the size of a subgraph, and each plotted point shows the number of dense subgraphs of this size. Remarkably, all the plots seem to indicate that the subgraph sizes follow the power law distribution—roughly speaking, the number $P(x)$ of dense subgraphs is a power function of the subgraph size x, in the form $P(x) \propto x^{\gamma}$ with $\gamma < 0$. This adds yet one more instance to the family of power laws previously discovered on social and information networks [44], [45], the most notable of which is the power law distribution of the vertex degrees. Each plot of Fig. 8 also shows a line that is the least squares fit to the plotted data in log-log scale. The slope of the line, which is essentially the exponent γ, is typically in the range from -3.5 to -1.5.

It is clear from our algorithms that the extracted dense components resulting from a larger $d_{\min}$ are all subgraphs of those resulting from a smaller $d_{\min}$. This effectively means that in the power law expression $P(x) \propto x^{\gamma}$, the exponent γ tends to decrease as the threshold $d_{\min}$ increases, since the extracted subgraphs become smaller and smaller. This can be seen from Fig. 8, where in general the fitted line becomes steep when $d_{\min}$ is increasing. Further, the total number of vertices that belong to the extracted subgraphs will naturally decrease. A plot (Fig. 9) indicates that this decrease looks linear.

D. *A Blog Network Example*

In this subsection we analyze the structure of a blog network `polblogs`. The data set, a network that connects bloggers of different political orientations, was originally constructed around the time of the 2004 U.S. presidential election, to study the interactions between the two groups: liberal and conservative [36]. The graph contains 1,490 vertices, among which the first 758 are liberal blogs, and the remaining 732 are conservative. An edge in the graph indicates the existence of citations between the two blogs. As can be seen from Figure 10(a), there are much denser links between blogs that hold the same political orientation than between those with different leanings.

We ran our algorithm on this graph by using different density thresholds. A typical result is shown in plot (b), where $d_{\min} = 0.4$. Indeed, for all the thresholds we tried, only two dense subgraphs (of size larger than 4) were identified. These two subgraphs perfectly correspond to the two politically oriented groups: The smaller subgraph (except for one vertex in the situation of low density thresholds) consists of conservative blogs, whereas the larger subgraph consists of liberal blogs. Hence, these two subsets of blogs are truly representative of the two groups.

It is observed that the density of the smaller subgraph is in general larger than that of the larger subgraph. One conclusion from this is that conservative blogs tend to make a larger number of citations to each other than liberal ones. This happens to be in agreement with the point made in [36] that “right-leaning (conservative) blogs have a denser structure of strong connections than the left (liberal)”, a result of a different analysis using the number of citations between different blogs. However, since the size of the liberal subgraph is much larger than that of the conservative (cf. plot (c)), an alternative conclusion is that more liberal blogs are willing to cite each other than conservative ones. This is somehow opposite to the dense citations in conservative blogs.

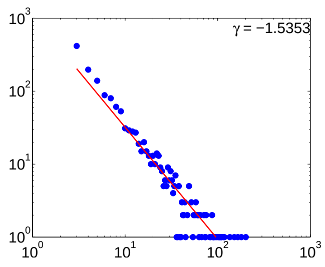
It is interesting to note here that plot (c) can suggest a way to select an “optimal” threshold $d_{\min}$. In this particular case, $d_{\min} = 0.4$ seems optimal, because beyond this point, the size of one of the subgraphs starts decreasing significantly, whereas there is no change when $d_{\min}$ grows from smaller values.

TABLE IV
RUNNING TIMES (UNIT: SECONDS) FOR THE GRAPHS IN TABLE III.

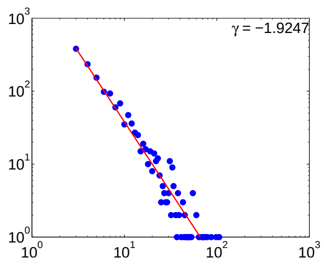
Graph	Type	$\lvert V\rvert$	$\lvert E\rvert$	τ	Similarity[a]	Sorting[b]	Hierarchy[c]	Density[d]	Extraction[e]
`polblogs`	directed	1,490	19,022	1	0.07	0.06	0.00	0.01	0.00
`yeast`	undirected	2,361	6,646	1	0.00	0.03	0.00	0.01	0.00
`Reuters911`	undirected	13,332	148,038	1	1.58	0.59	0.02	0.05	0.00
`foldoc`	directed	13,356	120,238	1	0.21	0.18	0.01	0.04	0.00
`hep`	directed	27,770	352,768	1	2.10	1.14	0.06	0.15	0.00
`epinions`	directed	49,288	487,182	3	3.86	2.04	0.12	0.17	0.02
`dictionary28`	undirected	52,652	89,038	1	0.22	0.11	0.04	0.08	0.01
`NDwww`	directed	325,729	1,469,679	30	13.98	42.07	2.46	0.67	0.07

Graph	Type	$\lvert V_1\rvert$	$\lvert V_2\rvert$	$\lvert E\rvert$	τ	Similarity[a]	Sorting[b]	Hierarchy[c]	Density[d]	Extraction[e]
`cmuSame`	bipartite	3,000	5,932	263,325	1	11.81	0.51	0.01	0.08	0.00
`cmuDiff`	bipartite	3,000	7,666	185,680	1	2.94	0.55	0.02	0.06	0.00
`cmuSim`	bipartite	3,000	10,083	288,989	1	5.46	1.03	0.01	0.10	0.00
`MovieLens`	bipartite	3,706	6,040	1,000,209	10	40.26	5.59	0.58	0.28	0.00
`newsgroup`	bipartite	18,774	61,188	2,435,219	1	140.32	11.15	0.21	0.87	0.02

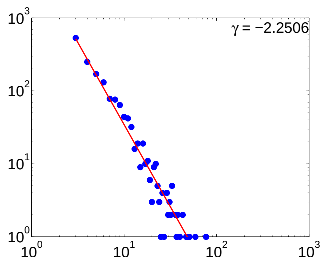
[a] The time to compute M or $\hat{M}$, including the modification of A in the bipartite graph case (cf. Sec. III-A).
[b] The time to sort $\tau \cdot \mathrm{nz}(A)$ nonzeros of M or $\hat{M}$ (cf. Sec. III-B).
[c] The time to construct the hierarchy T (cf. Sec. III-B).
[d] The time to compute the densities of all the subgraphs in the hierarchy (cf. Sec. III-C).
[e] The time to extract the dense subgraphs given a density threshold (cf. Sec. III-C).



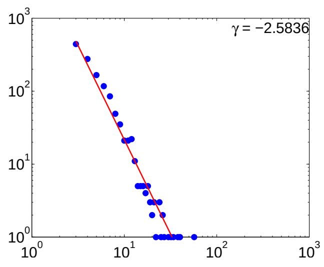
(a) `hep`: $d_{\min} = 0.2$.



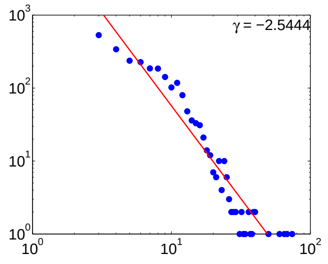
(b) `hep`: $d_{\min} = 0.4$.



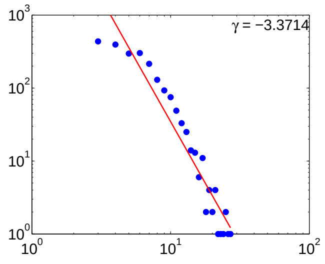
(c) `hep`: $d_{\min} = 0.6$.



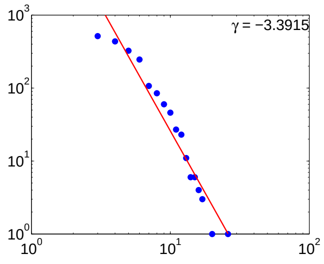
(d) `hep`: $d_{\min} = 0.8$.



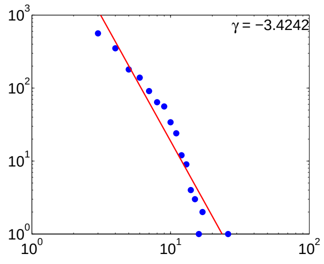
(e) `dictionary28`: $d_{\min} = 0.2$.



(f) `dictionary28`: $d_{\min} = 0.4$.



(g) `dictionary28`: $d_{\min} = 0.6$.



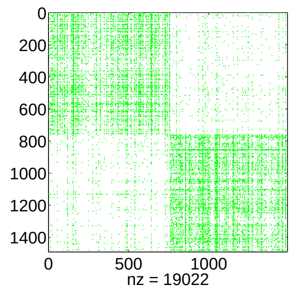
(h) `dictionary28`: $d_{\min} = 0.8$.

Fig. 8. Statistics of the extracted dense subgraphs for different density thresholds. The vertical axis is the number of subgraphs, and the horizontal axis is the subgraph cardinality. The plots are in log-log scale. Each red line is a least squares fit to the data, with its slope γ indicated at the upper right corner of each plot.

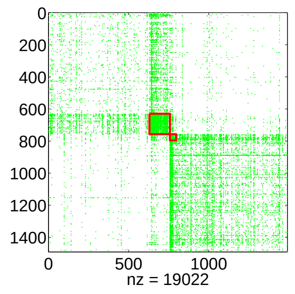
E. A Text Network Example

Words can be organized to form a network, where the structures of the relations between words can be exploited in order to analyze word usage and to understand linguistics. The data set `Reuters911` "is based on all stories released during 66 consecutive days by the news agency Reuters concerning the September 11 attack on the U.S., beginning at 9:00 AM EST 9/11/01." [38] It consists of 13,332 words from these news reports, and two words are connected if they appear in the same semantic unit (sentence here). By our technique (using a density threshold $d_{\min} = 0.5$), we extracted words that tend to be used together under such a context, such as those related to politics: `house of reps`, `senate`, `house`, `committee`, `capitol`, `hill`, `congressional`, `republican`, `senator`, `democrat`, those related to Arabic countries and names: `tunisia`, `yahya`, `benaissa`, `ben`, `habib`, `morocco`, `riziq`, `syihab`, and those related to the economic impacts: `market`, `stock`, `exchange`, `trade`, `wall street`.

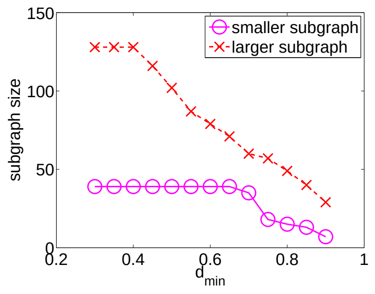
Perhaps the most important group of words (the largest extracted subgraph) is listed in Tab. V. They can be used as key words to summarize the 911 tragedy and the stories behind it.



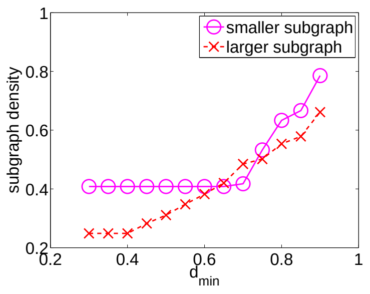
(a) Original graph.



(b) Two dense subgraphs.



(c) Subgraph size.



(d) Subgraph density.

Fig. 10. Dense subgraph extraction of a political blog network as shown in (a). Only two subgraphs (of size larger than 4) are identified for all the density thresholds experimented with. Plot (b) shows the two subgraphs (using $d_{\min} = 0.4$) in red boxes. Plots (c) and (d) show the changes in the sizes and the densities as $d_{\min}$ varies.

TABLE V
THE LARGEST GROUP OF WORDS THAT TEND TO APPEAR TOGETHER IN 911-RELATED NEWS REPORTS.

attack	united states	pres bush	official	people
washington	afghanistan	taliban	country	bin laden
afghan	american	kabul	al quaeda	force
troop	tuesday	wednesday	military	day
week	government	friday	thursday	monday
nation	support	pakistan	saudi-born	strike
new york	city	time	terrorism	terrorist
security	report	war	world	sunday
raid	network	new	air	alliance
opposition	capital	america	pakistani	militant
hijack	suicide	hijacker	aircraft	plane
flight	authority	leader	bomb	pentagon
kandahar	southern	stronghold	anthrax	case
bacterium	target	airport	possible	white house
group	information	campaign	operation	jet
fbi	letter	mail	test	dissident
deadly	month	part	threat	federal
tower	twin	110-story	world trade ctr	sept
state	saturday	islamic	muslim	11
man	member	fighter	agency	

F. A Bipartite Graph Example

Bipartite graph models are common in text mining, recommender systems, and other research fields. We show the `newsgroup` example where the dense subgraph extraction results can be interpreted as a partial co-clustering of the terms and the documents. Unlike existing co-clustering approaches [46]–[49] that return a complete clustering of the data matrix, our method returns only a subset of the entities where dense connections exist in each cluster.

The data set `newsgroup` (see Tab. III) is organized as a term-document matrix, where there are approximately 18,774 documents from 20 different newsgroups. The dictionary (number of terms) has size 61,188. The matrix represents a sparse graph where connections are drawn between two types of entities: terms and documents. We extracted dense subgraphs using the parameter $d_{\min}$ ranging from 0.1 to 0.9, and required that a subgraph should consist of at least 5 documents and 3 terms. To measure the clustering quality of the documents, we compute the entropy and the purity [50] of the document clusters. Fig. 11 shows the plot. It indicates that the document clusters are pure, especially when the density threshold is high. The plot also shows the total number of clustered documents. It varies from 10% to 30% of the whole document set. From the document clusters, we inspect the corresponding terms. We use the extraction results of $d_{\min} = 0.9$. In Tab. VI, we list the largest four term clusters, and the newsgroup to which they (or most of them) correspond. It can be seen that the words are very relevant to the topics of the newsgroups.

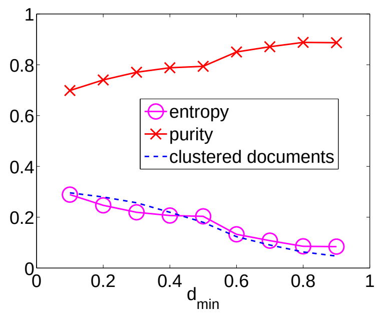


Fig. 11. Clustering quality of the documents in `newsgroup`: entropy and purity. "Clustered documents" is the percentage of documents that are clustered.

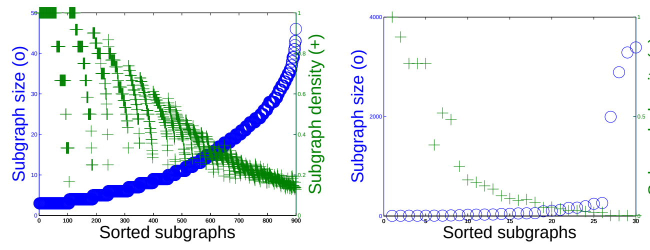
TABLE VI
THE LARGEST TERM CLUSTERS AND THE CORRESPONDING NEWSGROUPS.

talk.politics.mideast	talk.politics.guns	sci.crypt	misc.forsale
injuries	overwhelmed	transfering	cruising
comatose	conceded	betwen	barreling
boyhood	crudely	keyboards	liscence
pranks	detractors	numlock	occurance
devalued	outraged	micronics	reknowned
murderous	revocation	speedier	copious
municipalities	mailbombing	phantom	loper
⋮	⋮	⋮	⋮
(368 in total)	(29 in total)	(28 in total)	(28 in total)

G. Comparisons with the CNM Approach

We compare our approach with the one proposed by Clauset, Newman and Moore [10] (CNM). The CNM approach is in nature a close competitor to ours; it performs an agglomerative clustering on the graph vertices by greedily maximizing the modularity in each merge step. We would also like to include in the comparisons the divisive clustering approach [11] based on edge betweenness; however, the algorithm was very slow (at least with cubic time complexity) and did not terminate within hours even on a graph with around 10,000 vertices and 100,000 edges. Therefore, the approach [11] is not compared here. We demonstrate the comparisons using the dataset `foldoc`, which was extracted from the free on-line dictionary of computing (`http://foldoc.org/`). The vertices in the graph are terms related to computing, and there is an edge connecting two terms if one is used in the description/definition of the other.

Note first that it is difficult to find a single quantitative measure on the quality of the results. Criteria such as modularity, entropy or normalized mutual information are not appropriate for evaluating the subgraphs extracted from our algorithm, where the clustering is only partial and there lacks label information as the "ground truth". On the one hand, the subgraphs are trivially accurate in the sense that they are guaranteed to pass the density threshold. On the other hand, statistical properties may be considered when interpreting and evaluating the subgraphs. Using a density threshold $d_{\min} = 0.2$, we extracted 899 dense subgraphs, which contained in total 86.64% of the vertices of the whole network. The CNM approach divided the network into 30 clusters, where the four largest clusters contained 86.59% of the vertices, and the rest of the clusters were much smaller in size. Fig. 12 plots for each subgraph its size and its density. There is a clear trend for both approaches that larger subgraphs have smaller densities. The four largest subgraphs from the CNM approach are considered too large in the sense that their densities are very low and the theme, if any, represented by the terms in each subgraph is unclear. On the other hand, the subgraphs extracted from our approach are interpretable, which we will elaborate next.



(a) Our approach (b) CNM approach

Fig. 12. Subgraph sizes and densities.

By manual inspection, each of the small subgraphs, such as {refutable, regex, regexp}, {aliasing bug, precedence lossage, smash the stack, stale pointer bug}, {BCD, Binary Compatibility Standard, binaries, binary coded decimal, binary counter, binary file, packed decimal}, contains terms under a similar topic. In these examples, the topics are language patterns/expressions, programming errors, and data/number representation, respectively. Even for the large subgraphs (not listed here), we can also identify the themes. From the largest subgraph counted backwards, they represent advanced technology, device control and virtual machine, JAVA, Internet, etc. This interpretability reveals semantic structures of a network consisting of a large number of communities with moderate sizes, where members of a community tend to exhibit a common theme. On the contrary, the CNM approach tends to yield a small number of large clusters, which may be suitable for a network that can be divided into only a few categories.

V. Concluding Remarks

We have proposed a method to extract meaningful dense subgraphs from a given sparse graph (either undirected, directed, or bipartite). There are two major distinctions between the proposed method and previous ones that exploit complete clustering techniques. First, the output subgraphs are guaranteed to have high densities (above a certain prescribed threshold). Second, the number of clusters, which is in general difficult to estimate, is no longer a required parameter. The proposed algorithm is inspired by a matrix blocking technique which utilizes the cosine similarity of matrix columns. It effectively builds a hierarchy for the graph vertices, and computes a partial clustering for them. The real-life examples of Section IV indicate that the uses of the algorithm are flexible and the results are meaningful.

In the proposed algorithm, we introduced a density threshold parameter $d_{\min}$ to control the density of the output subgraphs. This parameter provides the flexibility needed to interactively explore the graph structure and the resulting communities. It can be tuned in real time, and results are easily visualized. The blog example in Sec. IV-D has shown the appeal of exploiting such a tunable parameter in understanding the extraction results.

The experiment in Sec. IV-C unraveled what appeared to be a new power law for large sparse graphs: the power law distribution of the dense subgraph sizes. It is still unclear if this interesting phenomenon is intrinsic to real-life complex systems. This newly discovered structure may have an influence on understanding the sizes of the communities in social networks.

A future avenue of research is to design algorithms to identify overlapping dense subgraphs. Many social and biological networks have shown empirically overlapping structures, where communities do not have a distinct borderline. The identification of such characters that connect different communities together may help better understand the network systems. We intend to explore how the algorithm proposed in this paper can be adapted for this task.

Acknowledgements

This research was supported by NSF grant DMS-0810938 and by the Minnesota Supercomputer Institute. The first author was supported in part by a University of Minnesota Doctoral Dissertation Fellowship. The authors would like to thank Arindam Banerjee for introducing us to the dense subgraph problem and in particular for bringing to our attention the importance of finding a partial clustering of the graph vertices.

References

[1] D. Gibson, J. Kleinberg, and P. Raghavan, "Inferring web communities from link topology," in *Proceedings of HYPERTEXT*, 1998.
[2] M. E. Newman, "Detecting community structure in networks," *Eur. Phys. J. B*, vol. 38, pp. 321–330, 2004.
[3] G. W. Flake, S. Lawrence, and C. L. Giles, "Efficient identification of web communities," in *Proceedings of ACM SIGKDD*, 2000.
[4] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *Proceedings of WWW*, 2008.
[5] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical review E*, vol. 74, no. 3, p. 036104, 2006.
[6] S. White and P. Smyth, "A spectral clustering approach to finding communities in graphs," in *Proceedings of SDM*, 2005.
[7] J. Abello, M. G. C. Resende, and S. Sudarsky, "Massive quasi-clique detection," in *Proceedings of LATIN*, 2002.
[8] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, no. 11–16, pp. 1481–1493, 1999.

[9] Y. Dourisboure, F. Geraci, and M. Pellegrini, "Extraction and classification of dense communities in the web," in *Proceedings of WWW*, 2007.

[10] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[11] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, 2004.

[12] K. Wakita and T. Tsurumi, "Finding community structure in mega-scale social networks," in *Proceedings of WWW*, 2007.

[13] J. P. Scott, *Social Network Analysis: A Handbook*, 2nd ed. Sage Publications Ltd, 2000.

[14] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *J. Machine Learning Research*, vol. 9, no. June, pp. 1981–2014, 2008.

[15] K. Nowicki and T. A. B. Snijders, "Estimation and prediction for stochastic blockstructures," *Journal of the American Statistical Association*, vol. 96, no. 455, pp. 1077–1087, 2001.

[16] K. Yu, S. Yu, and V. Tresp, "Soft clustering on graphs," in *Proceedings of NIPS*, 2005.

[17] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[18] I. Derényi, G. Palla, and T. Vicsek, "Clique percolation in random networks," *Physical Review Letters*, vol. 94, no. 16, 2005.

[19] L. Tang and H. Liu, "Graph mining applications to social network analysis," in *Managing and Mining Graph Data (Advances in Database Systems)*, C. C. Aggarwal and H. Wang, Eds., 2010.

[20] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," in *Managing and Mining Graph Data (Advances in Database Systems)*, C. C. Aggarwal and H. Wang, Eds., 2010.

[21] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, p. P09008, 2005.

[22] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[23] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 96–129, 1998.

[24] A. Abou-Rjeili and G. Karypis, "Multilevel algorithms for partitioning power-law graphs," in *Proceedings of IPDPS*, 2006.

[25] S. V. Dongen, "Graph clustering via a discrete uncoupling process," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 1, pp. 121–141, 2008.

[26] D. S. Hochbaum, "Polynomial time algorithms for ratio regions and a variant of normalized cut," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 32, no. 5, pp. 889–898, 2010.

[27] J. O'Neil and D. B. Szyld, "A block ordering method for sparse matrices," *SIAM J. Sci. Comput.*, vol. 11, no. 5, pp. 811–823, 1990.

[28] Y. Saad, "Finding exact and approximate block structures for ILU preconditioning," *SIAM J. Sci. Comput.*, vol. 24, no. 4, pp. 1107–1123, 2002.

[29] ——, *Iterative methods for sparse linear systems*, 2nd ed. SIAM, 2003.

[30] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons Ltd., 1990.

[31] J. Hopcroft, O. Khan, B. Kulis, and B. Selman, "Natural communities in large linked networks," in *Proceedings of ACM SIGKDD*, 2003.

[32] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of VLDB*, 2005.

[33] D. Eppstein, Z. Galil, and G. F. Italiano, "Dynamic graph algorithms," in *CRC Handbook of Algorithms and Theory of Computation*, 1997, ch. 22.

[34] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM J. Comput.*, vol. 13, no. 2, pp. 338–355, 1984.

[35] M. A. Bender and M. Farach-Colton, "The LCA problem revisited," in *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 2000, pp. 88–94.

[36] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 u.s. election: divided they blog," in *Proceedings of LinkKDD*, 2005.

[37] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen, "Topological structure analysis of the protein-protein interaction network in budding yeast," *Nucl. Acids Res.*, vol. 31, no. 9, pp. 2443–2450, 2003.

[38] S. R. Corman, T. Kuhn, R. D. McPhee, and K. J. Dooley, "Studying complex discursive systems: Centering resonance analysis of communication," *Human Communication Research*, vol. 28, no. 2, pp. 157–206, 2002.

[39] P. Massa and P. Avesani, "Trust-aware recommender systems," in *Proceedings of RecSys*, 2007.

[40] R. Albert, H. Jeong, and A. L. Barabási, "The diameter of the world wide web," *Nature*, vol. 401, pp. 130–131, 1999.

[41] A. Banerjee, I. Dhillon, J. Ghosh, and S. Sra, "Clustering on the unit hypersphere using von mises-fisher distributions," *J. Machine Learning Research*, vol. 6, pp. 1345–1382, 2005.

[42] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of SIGIR*, 1999.

[43] K. Lang, "Newsweeder: Learning to filter netnews," in *Proceedings of ICML*, 1995.

[44] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of SIGCOMM*, 1999.

[45] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[46] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of ACM SIGKDD*, 2001.

[47] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *Proceedings of ACM SIGKDD*, 2003.

[48] M. Rege, M. Dong, and F. Fotouhi, "Co-clustering documents and words using bipartite isoperimetric graph partitioning," in *Proceedings of ICDM*, 2006.

[49] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha, "A generalized maximum entropy approach to Bregman co-clustering and matrix approximation," *J. Machine Learning Research*, vol. 8, pp. 1919–1986, 2007.

[50] Y. Zhao and G. Karypis, "Empirical and theoretical comparisons of selected criterion functions for document clustering," *Machine Learning*, vol. 55, no. 3, pp. 311–331, 2004.