Hierarchical, Parameter-Free Community Discovery

Spiros Papadimitriou[1], Jimeng Sun[1], Christos Faloutsos[2], and Philip S. Yu[3]

[1] IBM T.J. Watson Research Center, Hawthorne, NY, USA
spapadim,jimeng@us.ibm.com

[2] Carnegie Mellon University, Pittsburgh, PA, USA
christos@cs.cmu.edu

[3] University of Illinois, Chicago, IL, USA
psyu@cs.uic.edu

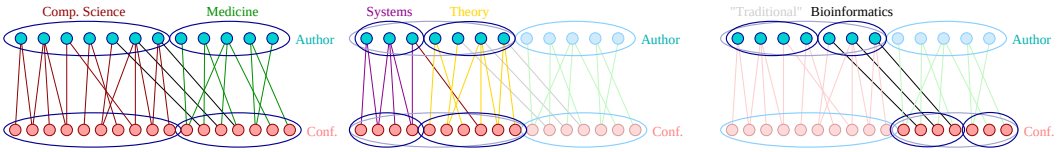
Abstract. Given a large bipartite graph (like document-term, or user-product graph), how can we find meaningful communities, quickly, and automatically? We propose to look for community hierarchies, with communities-within-communities. Our proposed method, the *Context-specific Cluster Tree (CCT)* finds such communities at multiple levels, with no user intervention, based on information theoretic principles (MDL). More specifically, it partitions the graph into progressively more refined subgraphs, allowing users to quickly navigate from the global, coarse structure of a graph to more focused and local patterns. As a fringe benefit, and also as an additional indication of its quality, it also achieves better compression than typical, non-hierarchical methods. We demonstrate its scalability and effectiveness on real, large graphs.

1 Introduction

Bipartite graphs (or, equivalently, sparse binary matrices) are natural representations of relations between two sets of nodes, namely source and destination nodes. Such large bipartite graphs arise naturally in many applications, like information retrieval (document-term graphs), collaborative filtering and recommendation systems (person-product graphs), social networks, and many more.

Graph mining aims at discovering the useful patterns hidden in the graphs. Various tools geared towards large graphs have been proposed in the literature. All of those techniques usually examine the graph at two extreme levels: 1) global, i.e., patterns present in the entire graph such as power law distribution on graphs [9], graph partitioning [16, 8, 4], community evolution [27, 25]; or, 2) local, i.e, patterns related to a subgraph such as center-piece graph [28], neighborhood formation [26], quasi-cliques [21].

In this paper, we aim to fill the gap between global and local patterns, by proposing a technique that allows users to effectively discover and explore communities in large graphs at multiple levels, starting from a global view and narrowing down to more local information. More specifically, we study ways to quickly and *automatically* construct a recursive community structure of a large bipartite graph at multiple levels, namely, a Context-specific Cluster Tree (CCT). The resulting CCT can identify relevant context-specific clusters. It also provides an efficient data summarization scheme and facilitates visualization of large graphs, which is a difficult, open problem itself [14, 13]. Intuitively, a *context* is a subgraph which is implicitly defined by a pair of source and destination



(a) First-level grouping (b) Second-level grouping (c) Alternative second-level grouping

Fig. 1. Hierarchy and context.

node groups (and, thus, includes exactly those edges that connect nodes of those groups)—see Definition 8. The entire graph and a single edge are the two extreme contexts, at the global and local level, respectively.

Our approach allows users to start from groups of nodes and edges at the global, coarse level and quickly focus on the appropriate context to discover more focused and fine-grained patterns. We shall illustrate the insight and intuition behind our proposed framework with an example. Consider a set of authors (blue nodes, top of Figure 1) and a set of conferences (red nodes, bottom of Figure 1), with edges indicating that the author published in that conference.

At first blush, one might discover a natural partitioning of the graph at the global level as follows:

- Node groups: Assume there are predominantly two groups of authors, computer scientists and medical researchers. Further assume there are two corresponding predominant conference groups. In matrix form, they correspond to the two row and column partitions, respectively, shown in Figure 2a.
- Contexts: The above node grouping leads to four contexts (i.e., edge groups, or subgraphs), one for each possible combination of the two node groups of each type (authors and conferences). In matrix form, they correspond to the four submatrices in Figure 1b. The dominant contexts are the two submatrices on the diagonal of Figure 2a corresponding to computer science (intersection of computer scientists and CS conferences) and medicine (intersection of of doctors and medical conferences), respectively.

This first-level decomposition already reveals much information about the structure of the data and answers the question: "given the mutual associations between *all* authors and *all* conferences, which are the groups of nodes that are most closely associated." Each of those group associations reveals a new context, which is a subgraph of the original graph. Thus, it can be likewise analyzed recursively, to reveal further contexts of finer granularity. In fact, if we stop at the first level and consider the computer science and medicine contexts, we may miss bioinformatics which will likely appear as associations between a subset of computer scientists and medical conferences (see Figure 2). To realize this intuition, we proceed to explain the two key concepts of hierarchy and context.

Hierarchy. Graphs often exhibit such community-within-community structure, leading to a natural recursive decomposition of their structure, which is a hierarchy. How can we find the appropriate number of levels, as well as the node groups within each level? These two questions bring in additional challenges to the design of the algorithm.

For example, let us consider the context induced by the "computer science" author and the "computer science" conference group (see Figure 1b, or the top-left part of Figure 2b). Performing a similar analysis as before, we may discover

additional structure in terms of node groups and contexts in the second level. The computer science field may be further subdivided into systems and theory authors, with a corresponding division in computer science conferences.

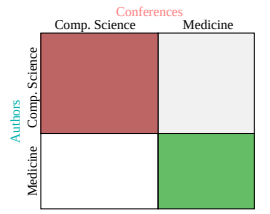
Context. In our example, the dominant contexts are those of "computer science" and "medicine," as explained above. However, there is nothing special about those "diagonal" contexts. In fact, we argue that one need also examine "off-diagonal" contexts. For example, the context defined by the intersection of "computer science" authors and "medical" conferences (see Figure 1c) may also be further partitioned into multiple lower-level contexts, with one of them corresponding to "bioinformatics".

In general, a particular choice of subgraph during the recursive decomposition consists of a pair of node groups and the context provided by the edges that associate them. Different contexts may reveal different aspects of the data. Taking this idea to its logical conclusion, the overall result is a rich hierarchy, CCT, that captures the graph structure at multiple levels.

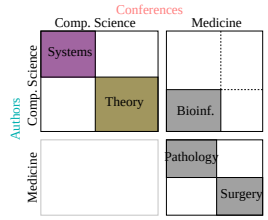
Our goal is to automatically find this hierarchy and allow users to quickly navigate it. For example, given a theoretician, there are multiple relevant contexts at different levels. Depending on the conferences, the most relevant context for her could be "computer theory" if the relevant conference is FOCS, SODA, or maybe "bioinformatics" if relevant conference is RECOMB or ISMB (see experiments for details). In general, the most relevant context can be automatically identified given the set of query nodes through a simple tree reversal.

Contributions. The main contributions of this paper are the following:

– We employ a parameter-free scheme based on minimum description language (MDL) that automatically finds the best context-specific cluster tree (CCT) to summarize the graph.



(a) First level



(b) Second level

Fig. 2. Adjacency matrix view (cf. Figure 1).

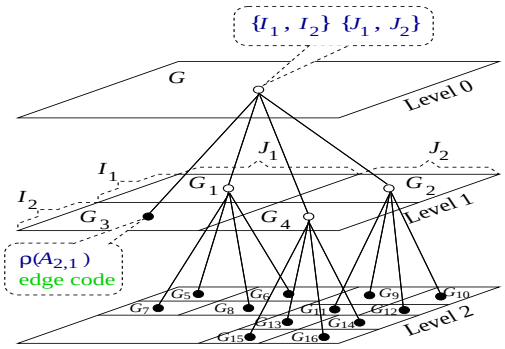


Fig. 3. Cluster tree (CCT) corresponding to Figure 2. Unfilled nodes correspond to non-leaves (subgraphs for which the partitioned model was best) and filled nodes correspond to leaves (subgraphs for which the random graph model was best). The two popups show examples of information represented at each type of node, with dark blue representing parts of the model and light green parts of the code, given the model.

– The method is linear on the number of edges, and thus scalable for large, possibly disk-resident, graphs.
– We provide a scheme for users to navigate from global, coarse structure to more focused and local patterns.

Because our method is based on sound, information theoretic principles, it also leads to better compression as a fringe benefit. Moreover, we develop a GUI prototype that allow users to visualize and explore large graphs in an intuitive manner. We demonstrate the efficiency and effectiveness of our framework on a number of datasets. In particular, a number of interesting clusters are identified in different levels.

The rest of the paper is organized as follows: Section 2 presents the necessary background and Section 3 introduces the fundamental definitions. Section 4 presents our proposed method and Section 5 evaluates it on a number of datasets. Finally, Section 6 briefly discusses related work and Section 7 concludes.

2 Background

In this section we give a brief overview of a practical formulation of the minimum description length (MDL) principle. For further information see, e.g., [7, 10]. Intuitively, the main idea behind MDL is the following: Let us assume that we have a family $\mathcal{M}$ of *models* with varying degrees of complexity. More complex models $M \in \mathcal{M}$ involve more parameters but, *given* these parameters (i.e., the model $M \in \mathcal{M}$), we can describe the observed data more concisely.

As a simple, concrete example, consider a binary sequence $A := [a(1), a(2), \ldots, a(n)]$ of n coin tosses. A simple model $M^{(1)}$ might consist of specifying the number h of heads. Given this model $M^{(1)} \equiv \{h/n\}$, we can encode the dataset A using $C(A|M^{(1)}) := nH(h/n)$ bits [22], where $H(\cdot)$ is the Shannon entropy function. However, in order to be fair, we should also include the number $C(M^{(1)})$ of bits to transmit the fraction h/n, which can be done using $\log^\star n$ bits for the denominator and $\lceil \log(n+1) \rceil$ bits for the numerator $h \in \{0, 1, \ldots, n\}$, for a total of $C(M^{(1)}) := \log^\star n + \lceil \log(n+1) \rceil$ bits.

Definition 1. (Code length and description complexity) $C(A|M^{(1)})$ *is* code length *for A, given the model $M^{(1)}$. $C(M^{(1)})$ is the* model description complexity *and $C(A, M^{(1)}) := C(A|M^{(1)}) + C(M^{(1)})$ is the* total code length.

A slightly more complex model might consist of segmenting the sequence in two pieces of length $n_1 \geq 1$ and $n_2 = n - n_1$ and describing each one independently. Let h_1 and h_2 be the number of heads in each segment. Then, to describe the model $M^{(2)} \equiv \{h_1/n_1, h_2/n_2\}$, we need $C(M^{(2)}) := \log^\star n + \lceil \log n \rceil + \lceil \log(n - n_1) \rceil + \lceil \log(n_1 + 1) \rceil + \lceil \log(n_2 + 1) \rceil$ bits. Given this information, we can describe the sequence using $C(A|M^{(2)}) := n_1 H(h_1/n_1) + n_2 H(h_2/n_2)$ bits.

Now, assume that our family of models is $\mathcal{M} := \{M^{(1)}, M^{(2)}\}$ and we wish to choose the "best" one for a particular sequence A. We will examine two sequences of length $n = 16$, both with 8 zeros and 8 ones, to illustrate the intuition.

Let $A_1 := \{0, 1, 0, 1, \cdots, 0, 1\}$, with alternating values. We have $C(A_1|M_1^{(1)}) = 16H(1/2) = 16$ and $C(M_1^{(1)}) = \log^\star 16 + \lceil \log(16+1) \rceil = 10 + 5 = 15$. However,

Symbol	Definition	Symbol	Definition
A	Binary adjacency matrix.	m_p, n_q	Dimensions of $A_{p,q}$.
m, n	Dimensions of A.	$\|A\|$	Number of elements $\|A\| := mn$.
k, ℓ	No. of source and dest. partitions.	$\rho(A)$	Edge density in $\rho(A) = e(A)/\|A\|$.
$A_{p,q}$	Submatrix for intersection of p-th source and q-th dest. partitions.	$H(\cdot)$	Shannon entropy function.
		$C(A)$	Codelength for A.

Table 1. Symbols and definitions.

for $M_1^{(2)}$ the best choice is $n_1 = 15$, with $C(A_1|M_1^{(2)}) \approx 15$ and $C(M_1^{(2)}) \approx 19$. The *total* code lengths are $C(A_1, M_1^{(1)}) \approx 16 + 15 = 31$ and $C(A_1, M_1^{(2)}) \approx 15 + 19 = 34$. Thus, based on total code length, the simpler model is better[4]. The more complex model may give us a lower code length, but that benefit is not enough to overcome the increase in description complexity: A_1 does not exhibit a pattern that can be exploited *by a two-segment model* to describe the data.

Let $A_2 := \{0, \cdots, 0, 1, \cdots, 1\}$ with all similar values contiguous. We have again $C(A_2|M_2^{(1)}) = 16$ and $C(M_2^{(1)}) = 15$. But, for $M_2^{(2)}$ the best choice is $n_1 = n_2 = 8$ so that $C(A_2|M_2^{(2)}) = 8H(0) + 8H(1) = 0$ and $C(M_2^{(2)}) \approx 24$. The *total* code lengths are $C(A_2, M_2^{(1)}) \approx 16 + 15 = 31$ and $C(A_2, M_2^{(2)}) \approx 0 + 24 = 24$. Thus, based on total code length, the two-segment model is better. Intuitively, it is clear that A_2 exhibits a pattern that can help reduce the total code length. This intuitive fact is precisely captured by the total code length.

3 CCT: Encoding and partitioning

We want to subdivide the adjacency matrix in tiles (or "contexts"), with possible reordering of rows and columns, and compress them, either as-is (if they are homogeneous enough) or by further subdividing. First, we formalize the problem and set the stage, by defining a lossless hierarchical encoding. As we shall see, the cluster-tree structure corresponds to the model, whereas the code for the data given the model is associated only with leaf nodes. This encoding allows us to apply MDL for automatically finding the desired progressive decomposition. In this section we define the codelength, assuming that a tree structure is given. Next, in Section 4, we present a practical algorithm to find the structure.

3.1 Problem definition

Assume we are given a set of m source nodes, $\mathcal{I} := \{1, 2, \ldots, m\}$ and a set of n destination nodes, $\mathcal{J} := \{1, 2, \ldots, n\}$. Each node pair (i, j), for $1 \le i \le m$ and $1 \le j \le n$, may be linked with an edge. Let $A = [a(i, j)]$ denote the corresponding $m \times n$ $(m, n \ge 1)$ binary adjacency matrix.

Definition 2 (Bipartite graph and subgraph). *The bipartite graph $\mathcal{G}$ is the triple $\mathcal{G} \equiv (\mathcal{I}, \mathcal{J}, A)$. A subgraph of this graph is a triple $\mathcal{G}' \equiv (\mathcal{I}', \mathcal{J}', A')$, where $\mathcal{I}' \subseteq \mathcal{I}$, $\mathcal{J}' \subseteq \mathcal{J}$ and $A' := [a(i', j')]$ for all $i' \in \mathcal{I}'$ and $j' \in \mathcal{J}'$.*

Our goal is to discover groups of edges that closely link groups of source nodes and destination nodes.

[4] The absolute codelengths are not important; the bit overhead compared to the straight transmission of A tends to zero, as n grows to infinity.

Definition 3 (Subgraph partitioning). *Given a graph $\mathcal{G} \equiv (\mathcal{I}, \mathcal{J}, A)$, we will partition it into a set of subgraphs $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_T\}$ such that their union equals the original graph $\mathcal{G}$.*

More specifically, we seek to decompose the original graph into a set of subgraphs, which should have the following properties:

- Connectedness: Each of the subgraphs should ideally be either fully connected or fully disconnected, i.e., it should be as homogeneous as possible.
- Flexible: The structure of the decomposition into subgraphs should be rich enough, without imposing too many constraints. On the other hand, it should lead to tractable and practical algorithms to find the decomposition.
- Progressive: The decomposition should allow users to navigate from global, coarse structure to more focused and local patterns, in the form of progressively more dense subgraphs.

Furthermore, we seek to automatically find such a decomposition, without requiring any parameters from the user. To that end, we employ MDL on an encoding of the bipartite adjacency matrix. The encoding we choose is hierarchical, so as to satisfy the last two properties.

3.2 Hierarchical encoding

In order to achieve the previously stated goals, we employ a top-down approach. Consider an $m \times n$ adjacency matrix A, which may correspond to any bipartite subgraph (including the entire graph). We proceed to explain how we can build a code for a *given* hierarchical partitioning.

Base case. Our first and simplest option is that no patterns are present in the data. In this case, we may safely model the data by simply assuming that each edge is independently drawn with probability $\rho(A)$, where $\rho(A)$ is the density of edges (or, of ones in the adjacency matrix A).

Definition 4 (Random graph model). *In this case, we may encode the entire matrix using*

$$C_0(A) := \lceil \log(|A| + 1) \rceil + \lceil |A| H(\rho(A)) \rceil \text{ bits.} \tag{1}$$

More specifically, we use $\lceil \log(|A|+1) \rceil$ bits to transmit $\rho(A)$ and finally $\lceil |A| H(\rho(A)) \rceil$ bits to transmit the individual edges. This assumes that we already know the graph size (i.e., m and n). For the initial graph $\mathcal{G}$, we may safely assume this. For its subgraphs, this information is provided by our model, as will be explained shortly.

Recursive case. The second option is to try to find joint groups of nodes and edges, as described before, which partition the original graph into subgraphs. Note that the partitioning (see Definition 3) is equivalent to a tiling of the adjacency matrix with T tiles, allowing for row and column reordering and, in the most general case, possibly overlapping tiles. Although we can allow arbitrary tilings, this leads to significant penalty in terms of complexity. Therefore, we impose certain constraints on the structure of the partitioning, so as to make the problem more tractable, while still allowing enough flexibility in the model to capture interesting patterns.

First, we require that the tiling is *exclusive* (i.e., no two tiles overlap) and *complete* (i.e., the tiles completely cover the entire adjacency matrix, without "gaps"). Next, we proceed to construct the tiling in a hierarchical fashion. We constrain the tiling to follow a checkerboard structure only within a single level of the hierarchy. The first-level decomposition of Figures 2a and 3 follows such a structure, consisting of $\mathcal{G}_1 = (\mathcal{I}_1, \mathcal{J}_1, A_{1,1})$, $\mathcal{G}_2 = (\mathcal{I}_1, \mathcal{J}_2, A_{1,2})$, $\mathcal{G}_3 = (\mathcal{I}_2, \mathcal{J}_1, A_{2,1})$, and $\mathcal{G}_4 = (\mathcal{I}_2, \mathcal{J}_2, A_{2,2})$, where $\mathcal{I}_1$ is the set of "computer science researchers" and $\mathcal{I}_2$ the set of "medical researchers" and similarly for the conference sets $\mathcal{J}_1$ and $\mathcal{J}_2$. Formally, the checkerboard structure means that set of source-destination group pairs, $\{(\mathcal{I}_1, \mathcal{J}_1), (\mathcal{I}_1, \mathcal{J}_2), (\mathcal{I}_2, \mathcal{J}_1), (\mathcal{I}_2, \mathcal{J}_2)\}$, can be written as a Cartesian product of individual sets of source and destination groups, $\{\mathcal{I}_1, \mathcal{I}_2\} \times \{\mathcal{J}_1, \mathcal{J}_2\}$.

In general, we can define a checkerboard decomposition into $T = k \cdot \ell$ subgraph tiles, using k source groups $\mathcal{I}_p$, for $1 \leq p \leq k$, and ℓ destination groups $\mathcal{J}_q$, for $1 \leq q \leq \ell$. We denote the sizes of $\mathcal{I}_p$ and $\mathcal{J}_q$ by $m_p := |\mathcal{I}_p|$ and $n_q := |\mathcal{J}_q|$, respectively, and the corresponding adjacency submatrices by $A_{p,q} := [a(\mathcal{I}_p, \mathcal{J}_q)]$, for $1 \leq p \leq k$ and $1 \leq q \leq \ell$.

Definition 5 (Partitioned graph model). *The cost of encoding the partitioned graph is*

$$C_1(A) := \lceil \log m \rceil + \lceil \log n \rceil + \left\lceil \log \binom{m}{m_1 \cdots m_k} \right\rceil + \\ + \left\lceil \log \binom{n}{n_1 \cdots n_\ell} \right\rceil + \sum_{p=1}^{k} \sum_{q=1}^{\ell} C(A_{p,q}). \quad (2)$$

We need $\lceil \log m \rceil$ bits to transmit k and $\lceil \log n \rceil$ bits to transmit ℓ. Furthermore, if we assume each mapping of m source nodes into k source groups equally likely, then we need $\lceil \log \binom{m}{m_1 \cdots m_k} \rceil$ bits to transmit the source partitioning $\{\mathcal{I}_1, \ldots, \mathcal{I}_k\}$, and similarly for the destination partitioning. Note that this partitioning implicitly determines the size $m_p \times n_q$ of each subgraph (which is assumed known by the random graph model). Finally, we need to recursively encode each of the $k \cdot \ell$ adjacency submatrices, one for each subgraph, which is represented by the last term in Equation (2).

Using Stirling's approximation $\ln n! \approx n \ln n - n$ and the fact that $\sum_i m_i = m$, we can easily derive that

$$\log \binom{m}{m_1 \cdots m_k} \approx m H\left(\tfrac{m_1}{m}, \ldots, \tfrac{m_k}{m}\right),$$

where $H(\cdot)$ denotes the Shannon entropy. Now we are ready to define the overall codelength cost $C(A)$.

Definition 6 (Total hierarchical codelength). *Given a hierarchical decomposition, the total codelength cost for transmitting the graph $(\mathcal{I}, \mathcal{J}, A)$ is*

$$C(A) := 1 + \min\{C_0(A), C_1(A)\}. \quad (3)$$

We choose the best of the two options, (i) pure random graph, or (ii) partitioned graph. Additionally, we need one bit to transmit which of these two options was the best. Ties are broken in favor of the simpler, random graph model. Note that the definition of the total cost is recursive, since $C(A_{p,q})$ appears in Equation (2).

Final result. To summarize, we have recursively built a hierarchical encoding based on a tiling of the adjacency matrix, each tile uniquely corresponding to a subgraph of the original bipartite graph. At each level of the hierarchy, we use checkerboard tiles. Each of those may be further subdivided in the same manner (such as all three tiles except the bottom left one from Figure 2a).

Definition 7 (Context-specific Cluster Tree). *The set of all subgraphs in the progressive, hierarchical decomposition consists the* context-specific cluster tree (CCT)*. The leaf nodes correspond to subgraphs for which the best choice is the random graph model. These subgraphs comprise the* leaf-level partitioning*. The* code *for the data given the model consists of the information for individual edges within subgraphs only at the leaf level.*

For example, in Figure 3, the root node would encode the partitioning $\{\mathcal{I}_1, \mathcal{I}_2\}$ and $\{\mathcal{J}_1, \mathcal{J}_2\}$; this is part of the model. The node corresponding to $\mathcal{G}_3 \equiv (\mathcal{I}_2, \mathcal{J}_1, A_{2,1})$ would encode the density $\rho(A_{2,1})$—which is also part of the model—and subsequently, the individual edges of $\mathcal{G}_3$ using entropy coding—which is part of the code given the model. In addition to the root $\mathcal{G}$, the CCT consists of all 16 nodes corresponding to subgraphs $\mathcal{G}_1$ through $\mathcal{G}_{16}$. The leaf-level partition consists of 13 graphs $\{\mathcal{G}_3, \mathcal{G}_5, \mathcal{G}_6, \ldots, \mathcal{G}_{16}\}$, which are represented by filled nodes.

It is clear from the construction that the leaf-level partitioning is also an exclusive and complete tiling, but with a richer structure than the per-level checkerboard tiles.

Finally, we can define the *context* for a set of nodes.

Definition 8 (Context). *Given as input a pair of source and destination node sets $(\mathcal{I}_i, \mathcal{J}_i)$ of interest, a* context *of $(\mathcal{I}_i, \mathcal{J}_i)$ is any pair $(\mathcal{I}_c, \mathcal{J}_c)$ such that $\mathcal{I}_i \subseteq \mathcal{I}_c$ and $\mathcal{J}_i \subseteq \mathcal{J}_c$.*

In other words, a context for $(\mathcal{I}_i, \mathcal{I}_j)$ is any subgraph of the original graph that fully includes $\mathcal{I}_i$ and $\mathcal{J}_i$. We will typically constrain $(\mathcal{I}_c, \mathcal{J}_c)$ to be only those pairs that appear in some node of our hierarchical decomposition. Given that constraint, we can give the next definition.

Definition 9 (Minimal hierarchical context). *The* minimal hierarchical context *among a set of contexts is the context $(\mathcal{I}_{mc}, J_{mc})$ such that no other context $(\mathcal{I}_c, \mathcal{J}_c)$ exists with $\mathcal{I}_{mc} \subseteq \mathcal{I}_c$ and $\mathcal{J}_{mc} \subseteq \mathcal{J}_c$*

Intuitively, the minimal hierarchical context is the deepest-level tree node in CCT that fully contains the input context. Note that, if $\mathcal{I}_i \neq \emptyset \neq \mathcal{J}_i$, then the minimal hierarchical context of $(\mathcal{I}_i, \mathcal{J}_i)$ is unique. If one of $\mathcal{I}_i$ or $\mathcal{J}_i$ is empty, then there may be multiple overlapping minimal contexts and, if both are empty, then all leaf nodes are trivially minimal contexts.

4 Finding the CCT

In the previous section we described the cost objective function that determines how "good" a given CCT partitioning is. However, if we don't know the partitioning we need a way to find it. The total codelength provides a yardstick to

Algorithm SHUFFLE:

Start with an arbitrary partitioning of the matrix A into k source partitions $\mathcal{I}_p^{(0)}$ and ℓ column partitions $\mathcal{J}_q^{(0)}$. Subsequently, at each iteration t perform the following steps:

1. For this step, we will hold destination partitions, i.e., $\mathcal{J}_q^{(t)}$, for all $1 \leq q \leq \ell$, fixed. We start with $\mathcal{I}_p^{(t+1)} := \mathcal{I}_p^{(t)}$ for all $1 \leq p \leq k$. Then, we consider each source node $i, 1 \leq i \leq n$ and move it into the p^*-th partition $\mathcal{I}_{p^*}^{(t+1)}$ so that the choice maximizes the "surrogate cost gain" $C_1'(A)$ of Equation (4).
2. Similar to step 1, but swapping destination nodes instead to find new partitions $\mathcal{J}_q^{(t+2)}$ for $1 \leq q \leq \ell$.
3. If there is no decrease in surrogate cost $C_1'(A)$, stop. Otherwise, set $t \leftarrow t + 2$, go to step 1, and iterate.

Fig. 4. Source and destination node partitioning, given the number of partitions.

Algorithm SPLIT:

Start with $k^0 = \ell^0 = 1$ and at each iteration τ:

1. Try to increase the number of source partitions, holding the number of destination partitions fixed. We choose to split the source partition p^* with maximum per-node entropy, i.e.,
$$p^* := \arg\max_{1 \leq p \leq k} \sum_{1 \leq q \leq \ell} |A_{p,q}| H(\rho(A_{p,q}))/m_p.$$
Increase the number of row partitions, $k^{\tau+1} = k^\tau + 1$ and construct a partitioning $\{\mathcal{I}_1^{(\tau+1)}, \ldots, \mathcal{I}_{k^{\tau+1}}^{(\tau+1)}\}$ by moving each node i of the partition $\mathcal{I}_{p^*}^{(\tau)}$ that will be split into the new source partition $\mathcal{I}_{k^{\tau+1}}^{(\tau+1)}$, if and only if this decreases the per-node entropy of the p^*-th partition.
2. Apply algorithm SHUFFLE with initial state $\{\mathcal{I}_p^{(\tau+1)} \mid 1 \leq p \leq k^{\tau+1}\}$ and $\{\mathcal{J}_p^{(\tau)} \mid 1 \leq p \leq \ell^\tau\}$, to find better assignments of nodes into partitions.
3. If there is no decrease in total cost, stop and return $(k, \ell) = (k^\tau, \ell^\tau)$ with corresponding partitions. Otherwise, set $\tau \leftarrow \tau + 1$ and continue.

4–6. Similar to steps 1–3, but trying to increase destination partitions instead.

Fig. 5. Algorithm to find number of source and destination partitions.

compare different hierarchical encodings with different number of partitions at each level of the hierarchy, but we need a practical search strategy to find such an encoding given only the initial graph.

In order to build a scalable and practical algorithm, we choose to employ a top-down strategy for building the hierarchy, rather than a bottom-up approach. Starting with the original graph, we try to find a good "checkerboard" tiling for the first level of the decomposition. Then, we fix this tiling and we recursively attempt the same procedure on each of the tiles.

However, there are two problems that need to be addressed. First, the recursive definition of Equation (2) is too expensive to evaluate for each possible assignment of nodes into partitions, so we use the following equation instead,

$$C_1'(A) := \lceil \log m \rceil + \lceil \log n \rceil + \left\lceil \log \binom{m}{m_1 \cdots m_k} \right\rceil + \\ + \left\lceil \log \binom{n}{n_1 \cdots n_\ell} \right\rceil + \sum_{p=1}^{k} \sum_{q=1}^{\ell} C_0(A_{p,q}), \quad (4)$$

where we have substituted C_0 for C in the summation at the end. This *surrogate cost* heuristic is fairly effective in practice, as we shall also see in the experiments.

Algorithm HIERARCHICAL:

1. Try SPLIT to find the best partitioned graph model.
2. Compare its codelength $C_1'(A)$ with that of the random graph model, $C_0(A)$.
3. If the partitioned graph model is better then, for each subgraph $(\mathcal{I}_p, \mathcal{J}_q, A_{p,q})$, for all $1 \leq p \leq k$ and $1 \leq q \leq \ell$, apply HIERARCHICAL recursively.

Fig. 6. Algorithm to find the cluster tree.

Even with this simplification, finding the optimal checkerboard tiling (i.e., assignment of nodes into partitions) is NP-hard [4], even if the number of tiles (or, equivalently, source and destination node partitions) is known. Additionally, we also seek the number of tiles. Therefore, we will employ an alternating minimization [4] scheme that converges towards a local minimum.

We recursively search for the best checkerboard partitions and stop when partitioned graph model is worse than the random graph model, which indicates the subgraph is sufficiently homogeneous. The search algorithm then proceeds in two steps: (i) an outer step, SPLIT, that attempts to progressively increase the number of source and destination partitions; and (ii) an inner step, SHUFFLE, that, given a fixed number of partitions, tries to find the best assignment of nodes into partitions. The pseudocode in Figures 4, 5, and 6 shows the steps of the overall process in detail.

Complexity. Algorithm SHUFFLE is linear with respect to the number of edges and the number of iterations. Algorithm SPLIT invokes SHUFFLE for each split, for a worst-case total of $2(k + \ell + 1)$ splits. For each level of the recursion in HIERARCHICAL, the total number of edges among all partitions of one level is at most equal to the number of edges in the original graph. Thus, the total time is proportional to the total number of edges, as well as the average leaf depth and number of partitions. Section 5 presents wall-clock time comparisons of the single-level and hierarchical algorithms.

5 Experimental evaluation

In this section we demonstrate our method on a number of real datasets. We compare against previous non-hierarchical/flat node partitioning schemes [4] and demonstrate that our proposed hierarchical graph decomposition provides significant benefits—in terms of revealing meaningful, interesting contexts and facilitating data exploration, summarization and compression—while still being scalable to large graphs and suitable for interactive visualization.

We implemented our algorithms in Matlab 7, with certain crucial parts (in particular, the main loop of SHUFFLE which iterates over all nodes) written in C as Matlab extensions (MEX). We have also developed a Matlab GUI[5] to facilitate navigation of the results and allow easy exploration of the clustering results.

The goal of the experiments is to show that CCT discovers intuitively meaningful subgraphs of various degrees of coarseness. We provide the evaluation from

[5] `http://www.cs.cmu.edu/~spapadim/dist/hcc/`

Theory authors				
Dhalia Malkhi	Nancy M. Amato	Yossi Matias	Monika R. Henzinger	Ronald L. Rivest
Joan Feigenbaum	**Robert Endre Tarjan**	Moni Naor	Michael T. Goodrich	David P. Dobkin
Thomas Lengauer	**Frank T. Leighton**	Jon M. Kleinberg	Ravi Kumar	Madhu Sudan
Nikhil Bansal	Eli Upfal	Lars Arge	Edith Cohen	Noga Alon
Richard Cole	Yishay Mansour	Randeep Bhatia	Sanjeev Khanna	Rajeev Motwani
Yonatan Aumann	Amit Kumar	Avi Widgerson	Arne Andersson	Vijaya Ramachandran
Micah Adler	Stefano Leonardi	Arnold Rosenberg	Gianfranco Bilardi	Ivan Hal Sudborough
Haim Kaplan	Jeffrey Scott Vitter	Cynthia Dwork	**Bhaskar Dasgupta**	Avrim Blum
Michael Mitzenmacher	**Mihalis Yannakakis**	**Anne Condon**	David R. Karger	Vwani P. Roychowdhuri
Richard E. Ladner	Wojciech Szpankowski	Amihood Amir	Sampath Kannan	**Tandy Warnow**

Table 2. All authors in the most dense "theory" context (see text). The conferences of this context are SODA, STOC, FOCS, and ICALP.

three aspects: navigation case-study, cluster quality, and method scalability. In particular we show that

- The hierarchical decomposition improves subgraph uniformity, with a reasonable number of subgraphs and an easy-to-navigate structure.
- The hierarchical decomposition achieves significantly lower total codelengths.
- Our approach is scalable to large datasets, with results progressively reported within very reasonable time.

5.1 Datasets

The first dataset, `DBLP`, consists of 8,539 source nodes representing authors that have published at least 20 papers and 3,092 destination nodes representing conferences. An edge represents that an author has published in the corresponding conference. The graph has 157,262 edges, or 0.60% non-zero entries in the adjacency matrix. The second dataset, `CLASSIC`, is from an information retrieval setting, with 3,893 source nodes representing documents and 4,303 destination nodes representing terms. The collection consists of papers from three different disciplines, medicine (MEDLINE), information retrieval (CISI) and aerodynamics (CRANFIELD), and has 176,347 edges, or 1.05% non-zero entries. The last dataset, `ENRON`, is from a social network setting, with 37,335 email addresses. Source nodes correspond to senders and destination nodes to recipients, with an edge representing the fact that the corresponding parties exchanged an email at some point in time. The graph has 367,660 edges, or 0.03% non-zero entries.

5.2 Navigating the results—case study

We focus our intuitive explanation of results on the `DBLP` dataset, due to space constraints and also better familiarity with the domain. The dataset consists of author-conference associations from DBLP. We have kept authors with at least 20 publications. In this setting, homogeneous subgraphs consist of authors that publish in similar venues. One should also keep in mind that most researchers have worked in more than one areas over the years and thus the venues they publish in differ over time.

We start our navigation by seeking the most specific theory context. For this purpose, we choose SODA, STOC and FOCS as representative of "theory"

Theory/bioinformatics authors				
Robert Endre Tarjan	Xin He	Francis Y.L. Chin	**Frank T. Leighton**	**Haim Kaplan**
Frank Hoffmann	**Bhaskar Dasgupta**	**Tandy Warnow**	**Mihalis Yannakakis**	**Anne Condon**
Tao Jiang	Christos H. Papadimitriou	**Michael Mitzenmacher**	Richard M. Karp	Piotr Berman

Table 3. Theory authors in most specific context w.r.t. RECOMB. Authors common with Table 2 are highlighted in bold.

(Rakesh Agrawal, SIGMOD) – 3rd level, 100%				
Authors (35)				Conf. (3)
Rakesh Agrawal	Joseph M. Hellerstein	Bruce G. Lindsay	Daniel M. Dias	SIGMOD
Peter J. Haas	Soumen Chakrabarti	Kenneth C. Sevcik	Luis Gravano	VLDB
Johannes Gehrke	Jim Gray	S. Muthukrishnan	Bettina Kemme	ICDE
Anastassia Ailamaki	Samuel Madden	S. Sheshadri	...	
(Hari Balakrishnan, SIGCOMM) – 4th level, 58%				
Authors (17)				Conf. (3)
Hari Balakrishnan	Ion Stoica	Srinivasan Sheshan	Leonard Kleinrock	SIGCOMM
M. Frans Kaashoek	Ramesh Govindan	Mario Gerla	J.J. Garcia-Luna-Aceves	MOBICOM
Christophe Diot	...			ICNP

Table 4. Pair contexts.

and we seek the densest leaf subgraph that contains at least these three conferences. This leaf is three levels deep in our decomposition. Table 2 shows the complete list of 50 authors in that subgraph, which indeed consists mostly of well-known theory researchers. Additionally, our method automatically included ICALP (Intl. Colloq. on Automata, Lang. and Programming) in the set of conferences for that subgraph. The density of this cluster is approximately 90%.

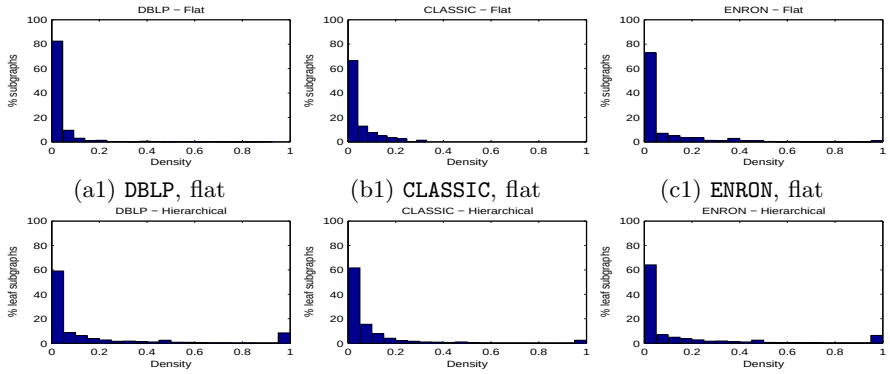
Next, we navigate up to the top-level subgraph that contains the same theory authors and seek the most specific context with respect to bioinformatics conferences. First, we chose RECOMB, which is more theory-oriented. Table 3 shows those 15 authors from that subgraph, which is also at the third level and has a density of about 40%. This leaf subgraph also includes the IEEE Conf. on Comp. Complexity, the Structure in Compl. Th. Conf. and CPM, as well as an additional 18 authors that have published in these conferences only. In general, the subgraph of theory people publishing in bioinformatics is far less dense than those subgraphs in their core areas, but still exhibits structure. Note that, since the ("theory", RECOMB) node is a sibling of the SODA-STOC-FOCS node, the list of authors in Table 3 is not a subset of the list in Table 2; authors common in both are highlighted in bold. Certain authors who have recently focused on bioinformatics are now included.

We also chose ISMB, another bioinformatics conference with less theoretical focus—hence it is in a different top-level partition than RECOMB. By navigating to the most dense leaf subgraph for the same set of theory authors, we quickly find that its density is merely 5%, with almost no theory people publishing there.

Finally, in Table 4 we show the most specific contexts for two author-conference pairs from different disciplines. We show a partial list of authors in the most specific context, due to space. The headings list the total number of author and conference nodes, as well as the level and density of the most specific subgraph.

First, from data management and mining we chose (Rakesh Agrawal, SIGMOD). Note that the the conference list automatically includes VLDB and ICDE, which are the other two main database conferences. The author list includes many previous collaborators or coworkers of Rakesh Agrawal, mostly senior, who have (co-)published in similar venues over time. Some junior people with similar publishing records are also included.

Next, from networking we chose (Hari Balakrishnan, SIGCOMM). The conference list automatically includes MOBICOM and ICNP (Intl. Conf. on Net. Protocols), as well as well-known networking researchers that have published in



(a1) `DBLP`, flat (b1) `CLASSIC`, flat (c1) `ENRON`, flat

(a2) `DBLP`, hierarchical (b2) `CLASSIC`, hierarchical (c2) `ENRON`, hierarchical

Fig. 7. Subgraph density distribution: CCT gives more high-density clusters close to 1.

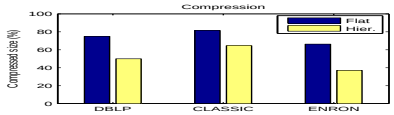
similar venues. Interestingly, INFOCOM is placed in a different subgraph from the highest level of the decomposition, since it has a much broader set of authors, who have also published in other areas.

5.3 Quantitative evaluation

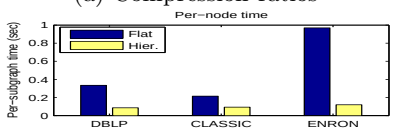
In this section we compare "Flat," which is the non-hierarchical, context-free approach in [4], and "Hier," which is our proposed CCT method, in terms of (i) subgraph uniformity, (ii) compression ratios, and (iii) computation time.

Subgraph uniformity. Figures 7(a1–c1) shows the distribution of subgraph edge densities ρ of the non-hierarchical approach, whereas Figures 7(a2–c2) show the same for the leaf-level subgraphs of our CCT decomposition. As expected, the leaf-level decomposition consists of more homogeneous subgraphs. The distribution of densities is clearly more concentrated towards either zero or one. The flat decomposition occasionally fails to produce any fully connected (or even close to fully connected) subgraphs. At the same time, the number of subgraphs in CCT is still fairly reasonable, typically at most 6–7% of the number of individual edges. The increased homogeneity of the subgraphs in conjunction with the appropriate complexity of the overall decomposition also leads to good compression, as we shall see later. Finally, as the level of the decomposition increases, subgraph sizes become small in comparison to the size of the original graph, at a rate that is roughly exponential with respect to the depth. In almost all cases, the average depth is close to three, with the maximum ranging from eight to nine. As we shall see later, this is significant to allow progressive reporting of results in interactive graph exploration.

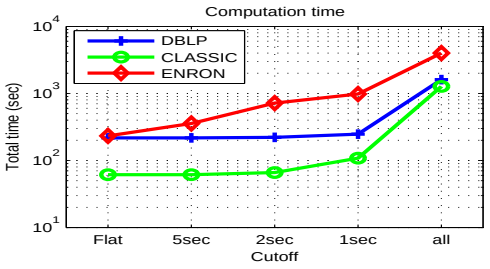
Compression. Figure 8a shows the compression achieved by the non-hierarchical approach and by CCT. We estimate the number of bits needed to store the original, "raw" matrix as $\lceil mnH(\rho(A)) \rceil$, i.e., we use the random graph model for the entire matrix. One can get very close to this estimate in practice by using techniques such as arithmetic coding [22]. Similarly, for the hierarchical decomposition we use the cost $C(A)$, from Equation (3). For the non-hierarchical approach, we use the cost formula from [4]. The figure shows the compression ratios for each method. It is clear from Figure 8a that CCT achieves significantly better compression, since it produces much more homogeneous subgraphs (see also Figure 7), while maintaining an appropriate number of partitions.



(a) Compression ratios



(b) Per-subgraph time



(c) Startup time

Fig. 8. (a) Compression ratios. CCT is significantly better, due to better balance between number of clusters and their homogeneity. (b,c) Wall-clock times. In (b) we see that, if users can tolerate a 1 sec navigation delay, the precomputation cost for our hierarchical approach (CCT) is almost as cheap as the context-free approach.

Scalability and progressive result reporting. Figures 8(b,c) shows measurements of wall-clock time for our prototype implementation in Matlab 7. The experiments were measured on a Pentium 4 running at 3GHz, with 2GB of memory. Figure 8b shows the total wall-clock time divided by the number of subgraphs produced by each method (previous non-hierarchical approach and CCT). As discussed before, subgraph sizes decrease quickly with respect to node depth. Thus, the processing time required to further decompose them decreases dramatically. Figure 8c shows the total wall-clock time if we were to compute: (i) just the first level of the decomposition, (ii) all nodes in the decomposition that require at least 5, 2, or 1 seconds, and (iii) all nodes at all levels of the decomposition. In an interactive graph exploration scheme, this effectively means that if we are willing to tolerate a delay of at most 5, 2 or 1 seconds when we wish to drill down a subgraph context, then the time required to pre-compute all other results would be equal to the corresponding y-axis value in Figure 8c. For example, if we are willing to tolerate at most 2 seconds "click lag," then `DBLP` requires 221 seconds (3.5 minutes) of pre-computation, `ENRON` 717 seconds (or, 12 minutes), versus 1609 seconds (27 minutes) and 3996 seconds (1 hour 7 minutes), respectively, for pre-computing everything.

6 Related work

We now survey related work beyond graph mining mentioned in Section 1 [9, 16, 27, 25, 28, 26, 21].

Biclustering. Biclustering/co-clustering [19] simultaneously clusters both rows and columns into coherent submatrices (biclusters). Cheng and Church [5] proposed a biclustering algorithm for gene expression data analysis, using a greedy algorithm that identifies one bicluster at a time by minimizing the sum squared residue. Cho et al. [6] use a similar measure of coherence but find all the biclusters simultaneously by alternating K-means. Information-theoretic Co-clustering [8] uses an alternating minimization algorithm for KL-divergence between the biclusters and the original matrix. The work in [4] formulates the biclustering problem as a binary matrix compression problem and also employs a local search scheme based on alternating minimization. However, it does not study hierarchical decomposition schemes or context-specific graph analysis and exploration. More recently, streaming extensions of biclustering has been proposed in [1, 25].

There are two main distinctions between our proposed method and existing work: 1) Most existing methods require a number of parameters to be set, such as number of biclusters or minimum support. 2) Existing methods are context-free approaches, which find biclusters global to the entire dataset, while our approach is context-specific and finds communities at multiple levels. Liu et al. [18] have leveraged the existing ontology for biclustering, which assumes the hierarchy is given, while our method automatically learns hierarchy from the data.

Hierarchical clustering. Hierarchical clustering builds a cluster hierarchy over the data points. The basic methods are agglomerative (bottom-up) or divisive (top-down) approaches through linkage metrics [12]. Following that spirit, a number of more sophisticated methods are developed, such as CURE that takes special care of outliers [11], CHAMELEON that relies on partitioning the k-NN graph of the data [15], and BIRCH that constructs a Cluster-Feature (CF) tree to achieve good scalability [30]. All these methods are one-dimensional, in the sense that all records (rows) are clustered based on all features (columns), while our proposed method clusters both records (rows) and features (columns) into coherent and context-specific groups. Another difference is that all these methods require a number of ad-hoc parameters, while our method is completely parameter-free.

Multilevel or multigrid methods for graph clustering [16, 2] and, more remotely related, local mesh refinement techniques pioneered in [3] also employ hierarchical schemes, but still require a few parameters (e.g., density thresholds). Finally, Yang et al develop techniques for 2D image compression via hierarchical, quadtree-like partitionings, which employ heuristics that are more powerful but still require a few parameters (e.g., for deciding when to stop recursion) and, more importantly, are less scalable [29].

Parameter-free mining. Recently, "parameter-free" as a desirable property has received more and more attention in many places. Keogh et al. [17] developed a simple and effective scheme for mining time-series data through compression. Actually, compression or Minimum Description Language (MDL) have become the workhorse of many parameter-free algorithms: frequent itemsets [24], biclustering [4, 23], time-evolving graph clustering [25], and spatial-clustering [20].

7 Conclusion

In this paper we develop the Context-specific Cluster Tree (CCT) for community exploration on large bipartite graphs. CCT has the following desirable properties: **(1) Parameter-free:** CCT is automatically constructed without any user intervention, using the MDL principle. **(2) Context-Specific:** Communities are detected at multiple levels and presented depending upon what contexts are being examined. **(3) Efficiency:** CCT construction is scalable to large graphs, and the resulting CCT can provide a compressed representation of the graph and facilitate visualization. Experiments showed that both space and computational efficiency are achieved in several large real graphs. Additionally, interesting context-specific clusters are identified in the DBLP graph. Future work could focus on parallelizing the CCT computation in order to speed up the construction.

8 References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *VLDB*, 2004.

[2] R. Bekkerman, R. El-Yaniv, and A. McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML*, 2005.

[3] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, 53, 1984.

[4] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-assocaiations. In *KDD*, 2004.

[5] Y. Cheng and G. M. Church. Biclustering of expression data. In *ISMB*, 2000.

[6] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. Minimum sum squared residue co-clustering of gene expression data. In *SDM*, 2004.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[8] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD*, 2003.

[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM*, pages 251–262, 1999.

[10] P. Grünwald. A tutorial introduction to the minimum description length principle. In *Advances in Min. Desc. Length: Theory and Applications*. MIT Press, 2005.

[11] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *SIGMOD*, 1998.

[12] J. Han and M. Kamber. *Data Mining*. Morgan Kaufmann, 2000.

[13] N. Henry and J.-D. Fekete. MatrixExplorer: a dual-representation system to explore social networks. In *InfoVis*, 2006.

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *InfoVis*, 2006.

[15] G. Karypis, E.-H. S. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8), 1999.

[16] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[17] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, pages 206–215, New York, NY, USA, 2004. ACM Press.

[18] J. Liu, W. Wang, and J. Yang. A framework for ontology-driven subspace clustering. In *KDD*, 2004.

[19] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM TCBB*, 1:24–45, 2004.

[20] S. Papadimitriou, A. Gionis, P. Tsaparas, R. A. Väisänen, C. Faloutsos, and H. Mannila. Parameter-free spatial data mining using mdl. In *ICDM*, 2005.

[21] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, 2005.

[22] J. Rissanen and G. G. Langdon Jr. Arithmetic coding. *IBM J. Res.*, 23(2), 1979.

[23] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *PNAS*, 104(18), 2007.

[24] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SDM*, 2006.

[25] J. Sun, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Graphscope: Parameter-free mining of large time-evolving graphs. In *KDD*, 2007.

[26] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Relevance search and anomaly detection in bipartite graphs. *SIGKDD Explorations*, 7, 2005.

[27] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *KDD*, 2006.

[28] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, 2006.

[29] Q. Yang, S. Lonardi, and A. Melkman. A compression-boosting transform for two-dimensional data. In *AAIM*, 2006.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, 1996.