AutoPart: Parameter-Free Graph Partitioning and Outlier Detection *

Deepayan Chakrabarti[1]

Carnegie Mellon University
deepay@cs.cmu.edu

Abstract. Graphs arise in numerous applications, such as the analysis of the Web, router networks, social networks, co-citation graphs, etc. Virtually all the popular methods for analyzing such graphs, for example, k-means clustering, METIS graph partitioning and SVD/PCA, require the user to specify various parameters such as the number of clusters, number of partitions and number of principal components. We propose a novel way to group nodes, using information-theoretic principles to choose both the number of such groups and the mapping from nodes to groups. Our algorithm is completely *parameter-free*, and also scales practically linearly with the problem size. Further, we propose novel algorithms which use this node group structure to get further insights into the data, by finding outliers and computing distances between groups. Finally, we present experiments on multiple synthetic and real-life datasets, where our methods give excellent, intuitive results.

1 Introduction - Motivation

Large, sparse graphs arise in many applications, under several guises. Consequently, because of their importance and prevalence, the problem of discovering structure in them has been widely studied in several domains, such as social networks, co-citation networks, ecological food webs, protein interaction graphs and many others. Such structure can be used for getting insights into the graph, for example, for detecting "communities".

Problem Description: A graph $G(V, E)$ has a set E of edges connecting any pair of nodes from a set V. Our definition includes both directed and undirected

* This material is based upon work supported by the National Science Foundation under Grants No. IIS-9817496, IIS-9988876, IIS-0083148, IIS-0113089, IIS-0209107 IIS-0205224 INT-0318547 SENSOR-0329549 EF-0331657IIS-0326322 by the Pennsylvania Infrastructure Technology Alliance (PITA) Grant No. 22-901-0001, and by the Defense Advanced Research Projects Agency under Contract No. N66001-00-1-8936. Additional funding was provided by donations from Intel, and by a gift from Northrop-Grumman Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

graphs. We want algorithms that discover structure in such datasets, and provide insights into them. Specifically, our goals are:

(G1) Clusters: "Similar" nodes should be grouped into "natural" clusters.
(G2) Outliers: Edges deviating from the overall structure should be tagged as outliers.
(G3) Inter-cluster Distances: For any pair of clusters, a measure of the "distance" between them should be defined.

In addition, the algorithms should have the following main properties:

(P1) Automatic: We want a principled and intuitive problem formulation, such that the user does not need to set *any* parameters.
(P2) Scalable: They should scale up for large, possibly disk resident graphs.
(P3) Incremental: They should allow online recomputation of results when new nodes and edges are added; this will allow the method to adapt to new incoming data from, say, web crawls.

In this paper, we propose algorithms to accomplish these objectives. Intuitively, we seek to group nodes so that the adjacency matrix is divided into rectangular/square regions as "similar" or "homogeneous" as possible. These regions of varying density would succinctly summarize the underlying structure of associations between nodes. In short, our method will take as input a matrix like in Figure 3(a) and produce Figure 3(g) as the output, without any human intervention.

The layout of the paper is as follows. In Section 2, we survey the related work. Subsequently, in Section 3, we formulate our data description model starting from first principles. Based on this, in Section 3.3 we outline a two-level framework to find homogeneous blocks in the adjacency matrices of graphs and develop an efficient, *parameter-free* algorithm to discover them. In Section 3.5, we use this structure to find outlier edges in the graph, and to calculate distances between node groups. In Section 4, we evaluate our algorithms, demonstrating good results on several real and synthetic datasets. Finally, we conclude in Section 5.

2 Survey

There has been quite a bit of work on graph partitioning. The prevailing methods are METIS [1] and spectral partitioning [2]. Both approaches have attracted a lot of interest and attention; however, both need the user to specify k, that is, the number of pieces the graph should be broken into. Moreover, they typically also require a measure of imbalance between the two pieces of each split. The *Markov Clustering* [3] method uses random walks, but is slow. Girvan and Newman [4] iteratively remove edges with the highest "stress" to eventually find disjoint communities, but the algorithm is again slow. Flake et al. [5] use the max-flow min-cut formulation to find communities around a seed node; however, the selection of seed nodes is not fully automatic.

Table 1. Table of symbols.

Symbol	Definition
D	Square binary adjacency matrix of a given graph
$d_{i,j}$	Entry in cell (i,j) of D; $d_{i,j} := 0$ or 1
n	Length of each side of D
k	Number of node groups
k^*	Optimal number of groups
$\mathcal{G}$	Node $\rightarrow$ group map
$\mathcal{G}_x$	Group corresponding to node x
$D_{i,j}$	Submatrix of links from group i to j
a_i	Number of nodes in group i
a_i, a_j	Dimensions of $D_{i,j}$
$n(D_{i,j})$	Number of elements in $D_{i,j}$; $n(D_{i,j}) := a_i a_j$
$w(D_{i,j})$	Weight of $D_{i,j}$ = number of "1"s in $D_{i,j}$
$P_{i,j}$	Density of "1"s in $D_{i,j}$; $P_{i,j} := w(D_{i,j})/n(D_{i,j})$
$H(p)$	Binary Shannon entropy function
$C(D_{i,j})$	Code cost for $D_{i,j}$
$T(D;k,\mathcal{G})$	Total cost for D

Remotely related are clustering techniques. Every row in the adjacency matrix can be envisioned as a multi-dimensional point. Several methods have been developed to cluster a cloud of n points in m dimensions, for example, k-means, k-harmonic means, CURE, BIRCH, Chameleon, LSI and others [6–8]. However, most current techniques require a user-given parameter, such as k for k-means. One solution called X-means [9] uses BIC to determine k. However, several of the clustering methods suffer from the dimensionality curse (like the ones that require a covariance matrix); others may not scale up for large datasets. Also, in our case, the points and their corresponding vectors are semantically related (each node occurs as a point *and* as a component of each vector); most clustering methods do not consider this. Other related work includes information-theoretic co-clustering (ITCC) [10]. However, the focus there is on lossy compression, whereas we employ a lossless MDL-based compression scheme. No MDL-like principle is yet known for lossy encoding, and hence, the number of clusters in ITCC cannot (yet) be automatically determined. Besides these, there has been work on conjunctive clustering [11] and community detection [12].

In conclusion, the above methods miss one or more of our prerequisite properties, typically not being automatic (P1). Next, we present our solution.

3 Proposed Method

Our goal is to find patterns in a large graph, with no user intervention, as shown in Figure 3. How should we decide the number of node groups k along with the assignments of nodes to their "proper" groups?

Compression as a guide: We introduce a novel approach and propose a general, intuitive model founded on compression, and more specifically, on the *MDL*

(Minimum Description Language) principle [13]. The idea is the following: the binary $n \times n$ matrix represents *associations* between the n nodes of the graph (corresponding to rows and columns in the adjacency matrix). If we mine this information properly, we could reorder the adjacency matrix so that "similar" nodes are grouped with each other. Then, the adjacency matrix would consist of homogeneous rectangular/square blocks of high(low) density, representing the fact that certain node groups have more(less) connections with other groups. To compress the matrix, we would prefer to have only a few blocks, each of them being very homogeneous. However, having more groups lets us create more homogeneous blocks (at the extreme, having n groups gives n^2 perfectly homogeneous blocks of size 1×1). Thus, the best compression scheme must achieve a tradeoff between these two factors, and this tradeoff point indicates the best number of node groups k. We accomplish this by a novel application of the overall MDL philosophy, where the compression costs are based on the number of bits required to transmit both the "summary" of the node groups, as well as each block given the groups. Thus, the user does not need to set any parameters; our algorithm chooses them so as to minimize these costs.

3.1 Compression Scheme for a Binary Matrix

Let $D = [d_{i,j}]$ denote an $n \times n$ adjacency matrix. Each graph node corresponds to one row and column in this matrix. We assume that $n \geq 1$. Let us index the rows and columns as $1, 2, \ldots, n$.

Let k denote the number of disjoint node groups. Let us index the groups as $1, 2, \ldots, k$. Let

$$\mathcal{G} : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, k\}$$

denote the assignments of nodes to groups. We can rearrange the underlying data matrix D so that all nodes corresponding to group 1 are listed first, followed by nodes in group 2, and so on. Such a rearrangement, implicitly, sub-divides the matrix D into k^2 smaller two-dimensional rectangular/square blocks, denoted by $D_{i,j}$, $i, j = 1, \ldots, k$. The more homogeneous these blocks, the better compression we can get, and so, the better the choice of $\mathcal{G}$. Table 3 lists the symbols used later.

We now describe a two-part code for the matrix D. The first part will be a *description complexity* involved in describing the blocks formed by $\mathcal{G}$ and the second part will be the actual *code* for the matrix given information about the blocks. A good choice of $\mathcal{G}$ will compress the matrix well, which will lead to low total encoding cost.

Description Cost: The description complexity (ie., information about the rectangular/square blocks) consists of the following terms:

1. Send the number of nodes n using $\log^\star(n)$ bits, where $\log^\star(x) = \log_2(x) + \log_2 \log_2(x) + \ldots$ with only the positive terms being retained [14]. This term is independent of $\mathcal{G}$ and k, and, hence, while useful for actual transmission of the data, will not figure in our framework.

2. Send the node permutations using $n\lceil\log n\rceil$ bits, respectively. Again, this term is also independent of $\mathcal{G}$ and k.
3. Send the number of groups k in $\log^\star k$ bits.
4. Send the number of nodes in each node group. Let us suppose that $a_1 \geq a_2 \geq \ldots \geq a_k \geq 1$. Compute $\bar{a}_i = \left(\sum_{t=i}^{k} a_t\right) - k + i$ for all $i = 1, \ldots, k-1$. Now, the desired quantities can be sent using $\sum_{i=1}^{k-1} \lceil\log \bar{a}_i\rceil$ bits
5. For each block $D_{i,j}$ $(i, j = 1, \ldots, k)$, send $w(D_{i,j})$, namely, the number of "1"s in $D_{i,j}$ using $\lceil\log(a_i a_j + 1)\rceil$ bits.

Code Cost: Suppose that the entire preamble specified above (containing information about the square and rectangular blocks) has been sent. We now transmit the actual matrix given this information. We can calculate the density $P_{i,j}$ of "1"s in $D_{i,j}$ using the description code above. The number of bits required to transmit $D_{i,j}$ is

$$\begin{aligned} C(D_{i,j}) &= n(D_{i,j}) H\big(P_{i,j}\big) \\ &= -w(D_{i,j}) \log\left(P_{i,j}\right) - [n(D_{i,j}) - w(D_{i,j})] \log(1 - P_{i,j}) \end{aligned} \tag{1}$$

where H is the binary Shannon entropy function, $n(D_{i,j}) = a_i a_j$, and all logarithms are base 2. Summing over all the $D_{i,j}$ submatrices:

$$\text{Code cost} = \sum_{i=1}^{k} \sum_{j=1}^{k} C(D_{i,j}) \tag{2}$$

Total Encoding Cost: We can now write the total cost for the matrix D, with respect to a given k and $\mathcal{G}$ as:

$$\boxed{T(D; k, \mathcal{G}) := \log^\star k + \sum_{i=1}^{k-1} \lceil\log \bar{a}_i\rceil + \sum_{i=1}^{k} \sum_{j=1}^{k} \lceil\log(a_i a_j + 1)\rceil + \sum_{i=1}^{k} \sum_{j=1}^{k} C(D_{i,j}) \quad (3)}$$

ignoring the costs $\log^\star(n)$ and $n\lceil\log n\rceil$ since they are independent of $\mathcal{G}$ and k.

3.2 Problem Formulation

We want an algorithm that can optimally choose k^* and $\mathcal{G}^*$ so as to minimize $T(D; k^\star, \mathcal{G}^\star)$. Typically, such problems are computationally hard, and hence, in this paper, we shall pursue feasible practical strategies. We solve the problem by a two-step iterative process: First, find a good node grouping $\mathcal{G}$ for a given number of node groups k; and second, search for the number of node groups k. For the former, we describe an iterative minimization algorithm to find a $\mathcal{G}$ that effectively finds a minimum, given a fixed number of node groups k. Then, we outline an effective heuristic strategy that searches over k to minimize the total cost $T(D; k, \mathcal{G})$.

3.3 Algorithms

In the previous section we established our goal: Among all possible values for k, and all possible node groups $\mathcal{G}$, pick an arrangement which reduces the total compression cost as much as possible, as MDL suggests (model plus data). Although theoretically pleasing, Equation 3 does not tell us *how* to go about finding the best arrangement - it can only pinpoint the best one, among several candidates! The question is: *how can we generate good candidates?*

We answer this question in two steps:

1. [`InnerLoop`] For a given k, find a good arrangement $\mathcal{G}$.
2. [`OuterLoop`] Efficiently search for the best k ($k = 1, 2, \ldots$).

Algorithm InnerLoop (Finding $\mathcal{G}$ given k):

1. Let t denote the iteration index. Initially, set $t = 0$. If no $\mathcal{G}(0)$ is provided, start with an arbitrary $\mathcal{G}(0)$ mapping nodes into k node groups. For this initial partition, compute the submatrices $D_{i,j}(t)$, and the corresponding distributions $P_{i,j}(t)$.
2. For every node x, splice the corresponding row into k parts $x_{row,1}, \ldots, x_{row,k}$ according to $\mathcal{G}(t)$ (i.e., $x_{row,1} = \{d_{x,u} | \mathcal{G}_u(t) = 1\}$ and so on). Similarly, splice the column into k parts $x_{col,1}, \ldots, x_{col,k}$. Compute the number of "1"s $w(x_{row,j})$ and $w(x_{col,j})$ ($j = 1 \ldots k$) for all these parts. Now, assign node x to node group $\mathcal{G}_x(t+1)$ such that

$$
\begin{aligned}
&\mathcal{G}_x\ (t+1) = \arg\min_{1 \le i \le k} \\
&\quad \left\{ \sum_{j=1}^{k} - \left[w(x_{row,j}) \log P_{i,j}(t) + (n(x_{row,j}) - w(x_{row,j})) \log(1 - P_{i,j}(t)) \right. \right. \\
&\quad \left. + \, w(x_{col,j}) \log P_{j,i}(t) + (n(x_{col,j}) - w(x_{col,j})) \log(1 - P_{j,i}(t)) \right] \\
&\quad + d_{x,x} \left[\log P_{i,\mathcal{G}_x(t)}(t) + \log P_{\mathcal{G}_x(t),i}(t) - \log P_{i,i}(t) \right] \\
&\quad \left. + \, (1 - d_{x,x}) \left[\log(1 - P_{i,\mathcal{G}_x(t)}(t)) + \log(1 - P_{\mathcal{G}_x(t),i}(t)) - \log(1 - P_{i,i}(t)) \right] \right\}
\end{aligned}
\tag{4}
$$

   where the first two lines denote the cost of shifting the row and column corresponding to node x to a new group, while the last two lines account for the "double-counting" of the cell $d_{x,x}$ in the adjacency matrix.
3. With respect to $\mathcal{G}(t+1)$, recompute the matrices $D_{i,j}^{t+1}$, and the corresponding distributions $P_{i,j}^{t+1}$.
4. If there is no decrease in total cost, stop; otherwise, set $t = t + 1$, go to step 2, and iterate.

Fig. 1. Algorithm `InnerLoop`

The `InnerLoop` algorithm iterates over several possible settings of $\mathcal{G}$ for the same number of node groups k. Each iteration improves (or maintains) the code cost, as stated in the theorem below.

Theorem 1. *After ach iteration of* `InnerLoop`*, the code cost decreases or remains the same. The proof is omitted for lack of space.*

The loop finishes when the total cost stops improving. Note that it is possible for some groups to be empty, but this is not a problem. The complexity of `InnerLoop` is $O(w(D) \cdot k \cdot I)$ where I is the number of iterations.

Algorithm OuterLoop (Finding k):

1. Let T denote the search iteration index. Start with $T = 0$ and $k(0) = 1$.
2. At iteration T, try to increase k: $k(T+1) = k(T)+1$. Split the node group r with maximum entropy per node, i.e.,

$$r := \underset{1 \leq i \leq k}{\arg\max} \sum_{1 \leq j \leq k} \frac{n(D_{i,j}) H\big(P_{i,j}\big) + n(D_{j,i}) H\big(P_{j,i}\big)}{a_i}$$

   Construct an initial label map $\mathcal{G}(T+1)$ as follows: For every node x that belongs to group r (i.e., for every $1 \leq x \leq n$ such that $\mathcal{G}_x(T) = r$), place it into the new group $k(T+1)$ (i.e., set $\mathcal{G}_x(T+1) = k(T+1)$) if and only if it decreases the per-node entropy of the group r, i.e., if and only if

$$\sum_{1 \leq j \leq k} \frac{n(D'_{r,j}) H\big(P'_{r,j}\big) + n(D'_{j,r}) H\big(P'_{j,r}\big)}{a_r - 1} < \sum_{1 \leq j \leq k} \frac{n(D_{r,j}) H\big(P_{r,j}\big) + n(D_{j,r}) H\big(P_{j,r}\big)}{a_r}$$

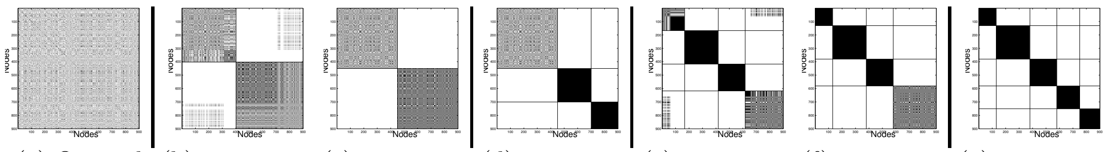
   where $D'_{r,j}$ is $D_{r,j}$ without node x. Otherwise let $\mathcal{G}_x(T+1) = r = \mathcal{G}_x(T)$. If we move node x to the new group, we also update $D_{r,j}$ and $D_{j,r}$ (for all $1 \leq j \leq k$) accordingly.
3. Run the `InnerLoop` algorithm with initial $\mathcal{G} = \mathcal{G}(T+1)$ and $k = k(T+1)$ to find a new node mapping $\mathcal{G}(T+1)$ and the corresponding total cost.
4. If there is no decrease in total cost, stop and return $k^* = k(T)$ and $\mathcal{G}^* = \mathcal{G}(T)$. Otherwise, set $T = T+1$ and continue.

Fig. 2. Algorithm `OuterLoop`

The `OuterLoop` algorithm tries to look for good values of k. It chooses the node group with the maximum entropy per node, and splits it into two groups. The nodes put into the new group are exactly the ones whose removal reduces the entropy per node in the original group. As shown below in Theorem 2, this split never decreases the code cost.

Theorem 2. *On splitting any node group, the code cost either decreases or remains the same. The proof is omitted due to lack of space.*

By Theorem 1, the same holds for `InnerLoop`. Therefore, the entire algorithm also decreases the code cost (Eq. 2). However, the description complexity evidently increases with k. We have found that, in practice, this search strategy performs very well. The `OuterLoop` algorithm is run k^* times, so the overall complexity of the search is $O(w(D)(k^*)^2 I)$. In practice, $I \leq 20$ is always enough.



(a) Original ($k = 1$) (b) `OuterLoop` ($k : 1 \to 2$) (c) `InnerLoop` ($k = 2$) (d) `OuterLoop` ($k : 2 \to 3$) (e) `OuterLoop` ($k : 3 \to 4$) (f) `InnerLoop` ($k = 4$) (g) `OuterLoop` ($k : 4 \to 5$)

Fig. 3. *Algorithm execution snapshots:* Starting with a randomly permuted "caveman" matrix (a), the algorithm applies `OuterLoop` and `InnerLoop` till the final structure (g) is revealed. We omit the `InnerLoop` results when they produce no improvement. Iterations of `OuterLoop` are separated by vertical lines for clarity.

Figure 3 shows an execution snapshot of the full algorithm on a randomly permuted "caveman" matrix (ie., a block diagonal matrix [15]) with Zipfian cave-sizes. `OuterLoop` increases the number of node groups while `InnerLoop` rearranges nodes between groups. No plots are shown when the `InnerLoop` does not decrease the total cost. The correct final result is shown in plot (g).

3.4 Online recomputations

When new nodes are obtained (such as from new crawls for a Web graph), we can put them into the node groups which minimize the increase in total encoding cost due to their addition. Based on the same principle, when new edges are found, the corresponding nodes can be reassigned to new node groups. The algorithm can then be run again with this initialization till it converges. Similar methods apply for node/edge deletions. Thus, new additions or deletions can be handled without full recomputations.

3.5 Using the block structure

Having found the underlying structure of a graph in the form of node groups, we can utilise this information to further mine the data. Again, we use our information-theoretic approach to answer several tough problems efficiently, using the node groupings found by the previous algorithms.

Outlier edges: Which edges between nodes are abnormal/suspicious? Intuitively, an outlier shows some deviation from normality, and so it should hurt attempts to compress data. Thus, an edge whose removal significantly reduces the total encoding cost is an outlier. Our algorithm is: Find the block where removal of an edge leads to the maximum immediate reduction in cost (that is, no iterations of the `InnerLoop` and `OuterLoop` algorithms are performed). All edges within that block contribute equally to the cost, and so all of them are considered outliers.

$$\boxed{\text{“Outlierness” of edge } (u, v) := T(D'; k, \mathcal{G}) - T(D; k, \mathcal{G}) \qquad (5)}$$

where $D' = D$ except that $d'_{u,v} = 0$. This can be used to *rank* the edges in terms of their "outlierness".

Table 2. Dataset characteristics.

Dataset	Num nodes	Num edges	Remarks
CAVE	900	170,800	Five "caves" with zipfian sizes
CAVE-Noisy	900	190,117	10% noise added to the above
NOISE	100	1,831	Pure white noise
EPINIONS	75,888	508,960	"Who-trusts-whom" data
DBLP	6,090	175,494	Coauthorship and cocitation data

"Distance" between node groups: How "close" are two node groups to each other? Following our information theory footing, we propose the following criterion: If two groups are "close", then combining the two into one group should not lead to a big increase in encoding cost. Based on this, we define "distance" between two groups as the relative increase in encoding cost if the two were merged into one:

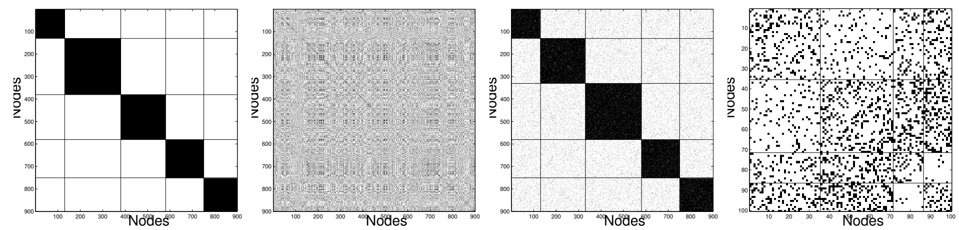
$$\boxed{Dist(i,j) := \frac{Cost(merged) - Cost(i) - Cost(j)}{Cost(i) + Cost(j)} \qquad (6)}$$

where only the nodes in groups i and j are used in computing costs. We experimented with other measures (such as the absolute increase in cost) but Eq 6 gave the best results. The cost of computing both outliers and distances between groups is independent of the number of non-zeros $w(D)$, and so both can be performed for large graphs.

4 Experiments

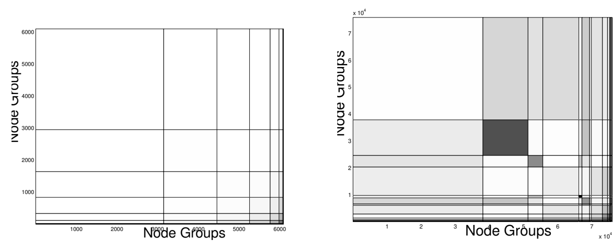
We did experiments to answer the following questions: (i) how good is the quality of the node groups found, (ii) how well do our algorithms find outlier edges, (iii) do our measures of "distances" between node groups make sense, and (iv) how well does our method scale up. All our experiments require no input parameters, which rules out other methods like METIS or spectral partitioning.

We used several datasets (see Table 3.5), both real and synthetic. The synthetic ones were: **(1)** `CAVE`, representing a social network of "cavemen" [15], that is, a block-diagonal matrix of variable-size blocks (or "caves"; members of a cave form a clique, and know only those from their own cave), **(2)** `CAVE-Noisy`, created by adding noise (10% of the number of non-zeros) , and **(3)** `NOISE`, with pure white noise. The real-world datasets are: **(4)** `EPINIONS`, a "who-trusts-whom" social graph of `www.epinions.com` users [16], and **(5)** `DBLP`, a graph obtained from `www.informatik.uni-trier.de/~ley/db`, with the nodes being authors in `SIGMOD, ICDE, VLDB, PODS` or `ICDT` (database conferences); two nodes are linked by an edge if the two authors have co-authored a paper or one has cited a paper by the other (thus, this graph is undirected). We performed experiments on other datasets too; the results were similar, and are not reported to save space. Our implementation was done in MATLAB (version 6.5 on Linux) using sparse matrices. The experiments were performed on an Intel Xeon 2.8GHz machine with 1GB RAM.



(a) CAVE (Final) (b) CAVE-Noisy (Original) (c) CAVE-Noisy (Final) (d) NOISE (Final)

Fig. 4. *Synthetic datasets*: (a) Our method gives the intuitively correct groups for CAVE (Figure 3(a) shows the original graph). (b,c) The results remain the same in spite of noise in CAVE-Noisy, showing the robustness of the algorithm. (d) The NOISE dataset shows 4 groups, which are explained by the patterns emerging due to randomness, such as the "almost-empty" and "more-dense" blocks.



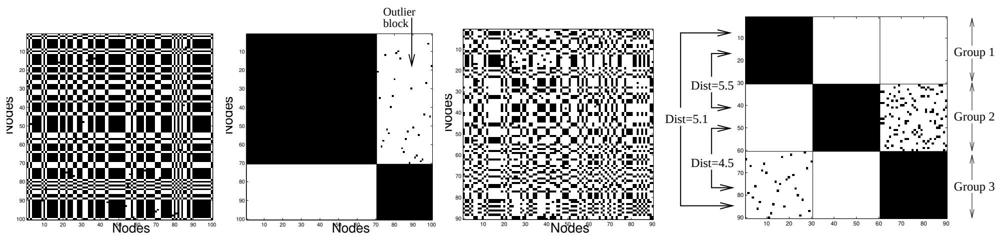
(a) DBLP ($k^* = 8$) (b) EPINIONS ($k^* = 19$)

Fig. 5. *Real datasets*: Shaded blocks are shown instead of the actual points; darker shades correspond denser blocks. The plots show how the algorithm has separated the graph into large but extremely sparse, and small but very dense groups. Most well-known database researchers show up in the dense regions of plot (a), as expected.

4.1 Quality

Results—synthetic data: Figure 4 shows the groupings found by our method on several synthetic datasets. For the noise-free `CAVE` matrix, we get exactly the intuitively correct groups (plot a). When noise is present (plot b), we still get the correct groups (plot c), demonstrating the robustness of our algorithm. Plot (d) shows 4 groups for the `NOISE` graph. This is expected; it is well known that spurious patterns emerge even when we have pure noise, and our algorithm finds blocks of clearly lower or higher density.

Results—real data: Figure 5 shows the groupings found on several real-world datasets. For the `DBLP` dataset, eight groups were found. Group 8 is comprised of only Michael Stonebraker, David DeWitt and Michael Carey; these are well-known people who have a lot of papers and citations. The other groups show decreasing number of connections but increasing sizes, with group 1 being the largest but having the lowest connectivity. Similarly, for the `EPINIONS` graph, we find a small dense "core" group which has very high connectivity, and then larger and less heavily-connected groupings. Thus, our method gives intuitive results for real-world graphs too.



(a) Outlier-test graph (b) Node groups and outliers (c) Distance-test graph (d) Node groups and distances

Fig. 6. *Outliers and group distances:* Plot (b) shows the node groups found for graph (a). Edges in the top-right block are correctly tagged as outliers. Plot (d) shows the node groups and group distances for graph (c). Groups 2 and 3 (having the most "bridges") are tagged as the closest groups. Similarly, groups 1 and 2 are the farthest.

4.2 Outlier edges

To test our algorithm for picking outliers, we use a synthetic dataset as in Figure 6(a). The node groups found are shown in 6(b). Our algorithm tags all edges whose removal would best compress the graph as outliers. Thus, all edges "across" the two groups are chosen as outliers under this principle (since all edges in a block contribute equally to the encoding cost), as shown in Figure 6(b). Thus, the intuitively correct outliers are found.

4.3 Distances between node groups

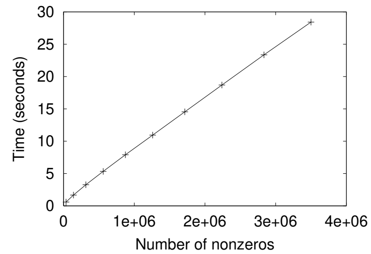
To test for node-group distances, we use the graph in 6(c) with 6(d) showing the structure found. The three caves have equal sizes but the number of "bridge" edges between groups varies. This is correctly picked up by our algorithm, which ranks groups with more "bridges" as being closer to each other. Thus, groups 2 and 3 are tagged as the "closest" groups, while groups 1 and 2 are "farthest".

4.4 Scalability

Figure 7 shows wall-clock times (in seconds) of our MATLAB implementation. The dataset is a "caveman" graph with 3 caves; the size of the graph and the number of edges in it are varied for the experiment, with the relative proportions of cave sizes being kept fixed. The execution time increases linearly with respect to the number of non-zeros, as expected from our order-of-complexity computation. Thus, our proposed method can scale to large graphs.

5 Conclusions

We considered the problem of finding the underlying structure in a graph. We introduced a novel approach and proposed a general, intuitive model founded on lossless compression and information-theoretic principles. Based on this model, we provided novel algorithms for finding node groups and outlier edges, as well as for computing distances between node groups, thus fulfilling all our goals (G1)-(G3) from Section 1. Our algorithms are fully automatic and parameter-free, scalable and allow online computations, achieving properties (P1)-(P3).



Time versus number of edges (nonzeros)

Fig. 7. *Scalability:* On a 3-cave graph, wall-clock execution time grows linearly with the number of edges. Thus, our method can scale to large graphs.

Finally, we evaluated our method on several real and synthetic datasets, where it produced excellent and intuitive results.

Acknowledgements: We would like to thank Dr. Faloutsos at CMU and the reviewers for their insightful comments and suggestions.

References

1. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: Proc. SC98. (1998) 1–13
2. Andrew Y.Ñg, Michael I.J̃ordan, Y.W.: On spectral clustering: Analysis and an algorithm. In: Proc. NIPS. (2001) 849–856
3. van Dongen, S.M.: Graph clustering by flow simulation. PhD thesis, Univesity of Utrecht (2000)
4. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. In: Proc. Natl. Acad. Sci. USA. Volume 99. (2002)
5. Flake, G.W., Lawrence, S., Giles, C.L.: Efficient identification of Web communities. In: KDD. (2000)
6. Zhang, B., Hsu, M., Dayal, U.: K-harmonic means - a spatial clustering algorithm with boosting. In: Proc. 1st TSDM. (2000) 31–45
7. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)
8. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. JASI **41** (1990) 391–407
9. Pelleg, D., Moore, A.: X-means: Extending K-means with efficient estimation of the number of clusters. In: Proc. 17th ICML. (2000) 727–734
10. Dhillon, I.S., Mallela, S., Modha, D.S.: Information-theoretic co-clustering. In: Proc. 9th KDD. (2003) 89–98
11. Mishra, N., Ron, D., Swaminathan, R.: On finding large conjunctive clusters. In: Proc. 16th COLT. (2003) 448–462
12. Reddy, P.K., Kitsuregawa, M.: An approach to relate the web communities through bipartite graphs. In: Proc. 2nd WISE. (2001) 302–310
13. Rissanen, J.: Modeling by shortest data description. Automatica **14** (1978) 465–471
14. Rissanen, J.: Universal prior for integers and estimation by minimum description length. Annals of Statistics **11** (1983) 416–431
15. Watts, D.J.: Small Worlds: The Dynamics of Networks between Order and Randomness. Princeton Univ. Press (1999)
16. Richardson, M., Domingos, P.: Mining knowledge-sharing sites for viral marketing. In: KDD, Edmonton, Canada (2002) 61–70