PaCK: Scalable Parameter-Free Clustering on K-Partite Graphs

Jingrui He* Hanghang Tong* Spiros Papadimitriou† Tina Eliassi-Rad‡
Christos Faloutsos* Jaime Carbonell*

Abstract

Given an author-paper-conference graph, how can we automatically find groups for author, paper and conference respectively. Existing work either (1) requires fine tuning of several parameters, or (2) can only be applied to bipartite graphs (e.g., author-paper graph, or paper-conference graph). To address this problem, in this paper, we propose **PaCK** for clustering such k-partite graphs. By optimizing an information-theoretic criterion, **PaCK** searches for the best number of clusters for each type of object and generates the corresponding clustering. The unique feature of **PaCK** over existing methods for clustering k-partite graphs lies in its parameter-free nature. Furthermore, it can be easily generalized to the cases where certain connectivity relations are expressed as tensors, e.g., time-evolving data. The proposed algorithm is scalable in the sense that it is linear with respect to the total number of edges in the graphs. We present the theoretical analysis as well as the experimental evaluations to demonstrate both its effectiveness and efficiency.

1 Introduction

Complex graphs that express various relationships among objects of different types are rapidly proliferating, largely due to the prevalence of the Web and the Internet. For example, the Resource Description Framework (RDF) [24] aims to express practically all multi-relational data in a standard, machine-understandable form. One of the Web's creators has even envisioned the Giant Global Graph (GGG) [5], which would capture and represent relationships across documents and networks.

In this paper we focus on a simpler but expressive subset of multi-relational data representations. We consider entities or objects of multiple types and allow any pair of types to be linked by a binary relationship. For example, in a publication corpus, papers (one object type) are associated with several other object types, e.g., authors, subject keywords, and publication venues.

Given such data, how can we find meaningful patterns and groups of objects, across different types? One approach might be to cluster the objects of each type independently of the rest. Traditional clustering techniques [13] are designed to group objects in an unlabeled data set such that objects within the same cluster are similar to each other, whereas objects from different clusters are dissimilar. Most clustering algorithms, such as k-means [9], spectral clustering [2] and information-theoretic clustering [11], focus on one-way clustering, i.e. clustering the objects according to their similarity based on the features.

However, for sparse relational data, co-clustering or bi-clustering techniques [18] *simultaneously* cluster objects of all types and typically produce results of better quality, by leveraging clusters along other types in the similarity measure. Most co-clustering algorithms focus on just two types of objects, typically viewed as either rows and columns of a matrix, or source and destination nodes of a bipartite graph. The information-theoretic co-clustering (ITCC) algorithm [8] was among the first in the machine learning community to address this problem. Follow up work includes [17], [15], and [3].

More recently, algorithms that generalize co-clustering to more than two object types have appeared, such as Consistent Bipartite Graph Co-partitioning (CBGC) [10], spectral relational clustering [16], and collective matrix factorization [20, 21]. These address some of the challenges in mining multi-relational data. However, despite their success, all of these methods require the user to provide several parameters, such as the number of clusters of each type, weights for different relations, etc.

We aim to provide a method that can also recover the number of clusters, by employing a model selection criterion based on lossless compression principles. Our starting point is the cross-associations [7] and Autopart [6] methods, both of which are parameter-free and provide the basic underpinnings for the model selection principles we shall employ. However, neither of them apply to multi-relational data. These pose additional challenges, which we address by introducing several new ideas, including exponential cluster splits, cluster merges, and multiple concurrent trials.

In this paper we propose **PaCK** to co-cluster k-partite graphs. Our main contributions in this paper are:

*Carnegie Mellon University
†IBM T.J. Watson Lab
‡Lawrence Livermore National Laboratory

- **PaCK** is parameter-free, by employing a simple but effective model selection criterion, which can recover the "true" cluster structure (when known).
- We carefully design a search procedure that can find a good approximate solution.
- Our algorithms are scalable to large datasets (linear on. number of edges).
- We generalize **PaCK** to tensors.

Extensive experiments on both real and synthetic datasets, include comparisons to several other methods, validate the effectiveness of **PaCK**.

The rest of the paper is organized as follows: Section 2 formulates the problem; Section 3 introduces the **PaCK** search procedure; and Section 4 presents experimental results; finally, Section 5 reviews related methods and Section 6 concludes.

2 Problem Formulation

In this section, we give the problem formulation of **PaCK**. Similar to cross-associations [7] and Autopart [6], **PaCK** tries to formulate the clustering problem as a compression problem. Unlike cross-associations [7] and Autopart [6], **PaCK**tries to compress a set of inter-correlated matrices collectively, as apposed to a single matrix in cross-associations and Autopart.

Given a set of inter-connected objects of different types, our goal is to find patterns based on the binary connectivity matrices in a parameter-free fashion, i.e. find the clusterings for different types of objects simultaneously so that after rearranging, the connectivity matrices will consist of homogeneous, rectangular regions of high or low density.

Generally speaking, to achieve this goal, we make use of the MDL (*Minimum Description Length*) principle to design a criterion, and try to minimize this criterion greedily. We describe the criterion in this section and deal with the search procedure in the next section.

2.1 Notation.

Given m types of data objects, $\mathcal{X}_1 = \{x_{11}, \ldots, x_{1n_1}\}, \ldots, \mathcal{X}_m = \{x_{m1}, \ldots, x_{mn_m}\}$, where $\mathcal{X}_i$ is the i^{th} object type, x_{ij} is the j^{th} object of the i^{th} type, and n_i is the number of objects from the i^{th} type, we are interested in collectively clustering $\mathcal{X}_1$ into k_1 disjoint clusters, ..., and $\mathcal{X}_m$ into k_m disjoint clusters. Let

$$\Phi_i : \{1, 2, \ldots, n_i\} \to \{1, 2, \ldots, k_i\},\ i = 1, \ldots, m$$

denote the assignments (i.e., mappings) of objects in $\mathcal{X}_i$ to the corresponding clusters.

Let D_{ij} denote the $n_i \times n_j$ binary connectivity matrix between object types $\mathcal{X}_i$ and $\mathcal{X}_j$, $i \neq j$, i.e., the element in the s^{th} row and t^{th} column of D_{ij} is 1 if and only if x_{is} is connected to x_{jt}.

To better understand the collective clustering, given the mappings $\Phi_i, i = 1, \ldots, m$, let us rearrange the connectivity matrix D_{ij} such that the objects within the same clusters are put together. In this way, the matrix D_{ij} is divided into smaller blocks, which are referred to as D_{ij}^{pq}, $p = 1, \ldots, k_i$ and $q = 1, \ldots, k_j$. Let the dimensions of D_{ij}^{pq} be (a_i^p, a_j^q). In other words, a_i^p is the number of objects from $\mathcal{X}_i$ that belong to cluster p, and a_j^q is the number of objects from $\mathcal{X}_j$ that belong to cluster q. Table 1 summarizes the notation used in this paper.

2.2 A Lossless Code for Connectivity Matrices.

Suppose that we are interested in transmitting the connectivity matrices D_{ij}, $i = 1, \ldots, m-1$ and $j = i+1, \ldots, m$. We are also given the mapping Φ_i that partitions the objects in $\mathcal{X}_i$ into k_i clusters, with none of them empty. Next, we introduce how to simultaneously code multiple matrices based on the above information.

2.2.1 Description Complexity. The first part is the description complexity of transmitting the connectivity matrices. It consists of the following parts:

1. Send the number of object types, i.e., $\log^*(m)$, where $\log^*$ is the universal code length for integers.[1] This term is independent of the collective clustering.
2. Send the number of objects of each type, i.e., $\sum_{i=1}^{m} \log^*(n_i)$. This term is also independent of the collective clustering.
3. Send the permutations of the objects of the same type so that the objects within the same clusters are put together, i.e., $\sum_{i=1}^{m} n_i \lceil \log k_i \rceil$.
4. Send the number of clusters using $\sum_{i=1}^{m} \log^* k_i$ bits.
5. Send the number of objects in each cluster. For object type $\mathcal{X}_i$, suppose that $a_i^1 \geq a_i^2 \geq \ldots \geq a_i^{k_i} \geq 1$. Compute
$$\bar{a}_i^p := (\sum_{t=p}^{k_i} a_i^t) - k_i + p$$
for $i = 1, \ldots, m$ and $p = 1, \ldots, k_i - 1$. So altogether the desired quantities can be sent using the following number of bits:
$$\sum_{i=1}^{m} \sum_{p=1}^{k_i - 1} \lceil \log \bar{a}_i^p \rceil$$
6. For each matrix block D_{ij}^{pq}, $i = 1, \ldots, m-1$, $j = i+1, \ldots, m$, $p = 1, \ldots, k_i$, and $q = 1, \ldots, k_j$, send the number of ones, using $\lceil \log(a_i^p a_j^q + 1) \rceil$ bits.

2.2.2 Code for the Matrix Blocks. In addition to the above information, we also need to transmit the matrix blocks D_{ij}^{pq}. For a single block D_{ij}^{pq}, we can model its elements as iid draws from a Bernoulli distribution with bias $P_{ij}^{pq} = n(D_{ij}^{pq}, 1)/(n(D_{ij}^{pq}, 1) + n(D_{ij}^{pq}, 0))$, where $n(D_{ij}^{pq}, 1)$ and $n(D_{ij}^{pq}, 0)$ are the numbers of ones and zeros

[1] $\log^*(x) \approx \log_2(x) + \log_2 \log_2(x) + \ldots$, where only the positive terms are retained and this is the optimal length, if we do not know the range of values for x beforehand.

Table 1: Notations

Symbol	Definition
m	Number of object types
$\mathcal{X}_i$	The i^{th} object type
n_i	Number of objects in $\mathcal{X}_i$
k_i	Number of clusters for $\mathcal{X}_i$
k_i^S	k_i in the S^{th} iteration step of **PaCK**
k_i^*	Optimal number of clusters for $\mathcal{X}_i$
Φ_i	Assignments/mappings of objects in $\mathcal{X}_i$ to the corresponding clusters
$\Phi_i(s)$	Φ_i in the s^{th} iteration step of **CCsearch**
Φ_i^S	Φ_i in the S^{th} iteration step of **PaCK**
Φ_i^*	Optimal assignments/mappings of objects in $\mathcal{X}_i$ to the corresponding clusters
D_{ij}	$n_i \times n_j$ binary connectivity matrix between object types $\mathcal{X}_i$ and $\mathcal{X}_j$
$D_{ij}\{t\}$	D_{ij} at time stamp t when the relationship between $\mathcal{X}_i$ and $\mathcal{X}_j$ is a tensor
a_i^p	Number of objects from $\mathcal{X}_i$ that belong to cluster p
D_{ij}^{pq}	Block of D_{ij} that corresponds to the p^{th} cluster in $\mathcal{X}_i$ and the q^{th} cluster in $\mathcal{X}_j$
$D_{ij}^{pq}(s)$	D_{ij}^{pq} in the s^{th} iteration step of **CCsearch**
$n(A,u)$	Number of elements in the matrix/vector A that are equal to u, $u = 0, 1$
$n(A)$	Total number of elements in the matrix/vector A
P_{ij}^{pq}	Proportion of 1s in D_{ij}^{pq}
$P_{ij}^{pq}(s)$	P_{ij}^{pq} in the s^{th} iteration step of **CCsearch**
$H(P_{ij}^{pq})$	Binary Shannon entropy function with respect to P_{ij}^{pq}
C_{ij}^{pq}	Coding length required to transmit the block D_{ij}^{pq} using arithmetic coding
$C_{ij}^{pq}(s)$	C_{ij}^{pq} in the s^{th} iteration step of **CCsearch**
$T_D(\Phi_1, \ldots, \Phi_m)$	Total coding cost with respect to the mappings $\Phi_1, \ldots, \Phi_m$

in D_{ij}^{pq}. Therefore, the number of bits required to transmit this block using arithmetic coding is

$$\begin{aligned} C_{ij}^{pq} &= C(D_{ij}^{pq}) := n(D_{ij}^{pq}) H(P_{ij}^{pq}) \\ &= -n(D_{ij}^{pq}, 1) \log(P_{ij}^{pq}) - n(D_{ij}^{pq}, 0) \log(1 - P_{ij}^{pq}) \end{aligned}$$

where $n(D_{ij}^{pq}) = n(D_{ij}^{pq}, 1) + n(D_{ij}^{pq}, 0)$, and H is the binary Shannon entropy function.

2.2.3 Total Coding Cost. Based on the above discussion, the total coding cost for the connectivity matrices D_{ij}, $i = 1, \ldots, m-1$, $j = i+1, \ldots, m$ with respect to the given mappings Φ_i, $i = 1, \ldots, m$ is as follows.

$$\begin{aligned} T_D(\Phi_1, \ldots, \Phi_m) &:= \sum_{i=1}^{m} n_i \lceil \log k_i \rceil + \sum_{i=1}^{m} \log^* k_i \\ &+ \sum_{i=1}^{m} \sum_{p=1}^{k_i - 1} \lceil \log \bar{a}_i^p \rceil + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{p=1}^{k_i} \sum_{q=1}^{k_j} \lceil \log(a_i^p a_j^q + 1) \rceil \\ &+ \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{p=1}^{k_i} \sum_{q=1}^{k_j} C_{ij}^{pq} \end{aligned} \tag{2.1}$$

Note that we ignore the costs $\log^*(m)$ and $\sum_{i=1}^{m} \log^*(n_i)$, since they do not depend on the collective clustering.

3 Search Procedure in PaCK

The optimal collective clustering corresponds to the number of clusters k_i^* and the mapping Φ_i^* for object type $\mathcal{X}_i$ such that the total coding cost $T_D(\Phi_1^*, \ldots, \Phi_m^*)$ is minimized. This problem is NP-hard[2]. Therefore, we have designed a greedy algorithm to minimize the total coding cost (Equation (2.1)). Specifically, to determine the optimal collective clustering, we must set the number of clusters for each object type, and then find the optimal mappings. These two components correspond to the two major steps in **PaCK**: (1) finding a good collective clustering given the number of clusters of each objective type; and (2) searching for the optimal number of clusters.

In this section, we first describe these two steps respectively, followed by computational complexity analysis for the proposed **PaCK**.

3.1 CCsearch

In **CCsearch** step, we are given the values of $k_1, \ldots, k_m$, and want to find a set of mappings $\Phi_1, \ldots, \Phi_m$ that minimizes the total coding cost (Equation (2.1)). Note that in this case, the first two terms in Equation (2.1) are fixed, and only the remaining three terms $\sum_{i=1}^{m} \sum_{p=1}^{k_i - 1} \lceil \log \bar{a}_i^p \rceil$, $\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{p=1}^{k_i} \sum_{q=1}^{k_j} \lceil \log(a_i^p a_j^q + 1) \rceil$ and $\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \sum_{p=1}^{k_i} \sum_{q=1}^{k_j} C_{ij}^{pq}$ depend on the mappings. Based on our experiments, in the regions where

[2] It is NP-hard since even allowing only column re-ordering for a single connectivity matrix, a reduction to the TSP problem can be found [14].

CCsearch searches for the optimal mappings given the numbers of clusters, the code for transmitting the matrix blocks dominates the total coding cost. Therefore, in **CCsearch**, we aim to minimize the following criterion:

$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{p=1}^{k_i}\sum_{q=1}^{k_j} C_{ij}^{pq} \tag{3.2}$$

CCsearch (Alg. 1) is an intuitive and efficient alternating minimization algorithm that yields a local minimum of Equation (3.2).

Note that **CCsearch** is essentially a Kmeans-style algorithm [19]: it alternates between finding the cluster centroid and assigning objects to the 'closest' cluster, except that in each iteration step, the 'features' (i.e., clusterings of other object types) of an object may change. This is different from the sequential clustering algorithm proposed in [22] where the cluster membership of only one object may change in each iteration.

The correctness of **CCsearch** is given by theorem 3.1.

THEOREM 3.1. *For $s \geq 1$*

$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{p=1}^{k_i}\sum_{q=1}^{k_j} C_{ij}^{pq}(s) \geq \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{p=1}^{k_i}\sum_{q=1}^{k_j} C_{ij}^{pq}(s+1)$$

where $C_{ij}^{pq}(s)$ is C_{ij}^{pq} in the s^{th} iteration step of **CCsearch**. *In other words,* **CCsearch** *never increases the objective function (Equation* (3.2)*).*

Proof. Omitted for brevity. ■

3.2 Cluster Number Search.

The second part of **PaCK** is an algorithm to look for good values of k_i (i.e., the cluster numbers for each object type). Here is the basic idea of **PaCK**: we start with small values of k_i, progressively increase them, and find the best Φ_i using **CCsearch**. We use two strategies to split the current clusters to increase the cluster number: the linear split (step 14) and exponential split (step 12). In order to escape the local minimum, we also allow merging two existing clusters into a bigger one (steps 26-36). Finally, in order to find a good local minimum, we propose to run **PaCK** multiple times and choose the one with the lowest coding cost. This step can be easily paralleled and therefore almost does not increase the overall running time. Note that unlike the outer loop of cross-associations [7], **PaCK** additionally performs the exponential split, merge operation and multiple concurrent trials over m ($m \geq 2$) types of objects. As we will show in the experimental section, these additional operations (exponential split, merge, and multiple concurrent trials) will significantly improve the search quality. The complete **PaCK** algorithm is summarized in Alg. 2.

In Alg. 2, if the split operation in the previous iteration for a given type l is not successful, we will try to split it

Algorithm 1 CCsearch

Input: The connectivity matrices $D_{i,j}$, $i, j = 1, ..., m$; the cluster numbers k_i, $i = 1, ..., m$ for each type.
Output: The cluster assignment Φ_i for each object type.

1: Let s denote the iteration index and set $s = 0$.
2: Initialize collective clustering $\Phi_1(s), \ldots, \Phi_m(s)$.
3: Set $\Phi_i'(s) = \Phi_i(s)$, $i = 1, \ldots, m$.
4: **while** true **do**
5: ### alternate among different types of objects
6: **for** $l = 1, \ldots, m$ **do**
7: ### try to update the clustering for type l
8: **for** $i = 1, \ldots, m-1$, $j = i+1, \ldots, m$, $p = 1, \ldots, k_i$, $q = 1, \ldots, k_j$ **do**
9: Compute the matrix blocks $D_{ij}^{pq}(s)$ and the corresponding bias $P_{ij}^{pq}(s)$ based on $\Phi_1'(s), \ldots, \Phi_m'(s)$.
10: **end for**
11: Hold the mapping $\Phi_l'(s)$ for object type $\mathcal{X}_l$. Concatenate all the matrices $D_{lj}, j \neq l$, to form a single matrix D_l, which is an $n_l \times \sum_{j \neq l} n_j$ matrix.
12: **for** each row x of D_l **do**
13: Split it into $\sum_{j \neq l} k_j$ parts, each corresponding to one cluster in $\Phi_j'(s)$, $j \neq l$.
14: Let the $\sum_{j \neq l} k_j$ parts found in step 11 be $x^{11}, \ldots, x^{1k_1}, \ldots, x^{(l-1)1}, \ldots, x^{(l-1)k_{l-1}}, x^{(l+1)1}, \ldots, x^{(l+1)k_{l+1}}, \ldots, x^{m1}, \ldots, x^{mk_m}$.
15: **end for**
16: **for** each of the $\sum_{j \neq l} k_j$ parts **do**
17: Compute $n(x^{jq}, u), u = 0, 1, j \neq l, q = 1, \ldots, k_j$, which is the number of elements in x^{jq} that are equal to u.
18: **end for**
19: Define $\Phi_l'(s)$ such that the cluster it assigns to row x satisfies the following condition: for all $1 \leq p \leq k_l, a + b \leq c + d$, where
$a = \sum_{j \neq l} \sum_{q=1}^{k_j} [n(x^{jq}, 1) \log \frac{1}{P_{lj}^{\Phi_l'(s)q}(s)}]$
$b = \sum_{j \neq l} \sum_{q=1}^{k_j} [n(x^{jq}, 0) \log \frac{1}{1 - P_{lj}^{\Phi_l'(s)q}(s)}]$
$c = \sum_{j \neq l} \sum_{q=1}^{k_j} [n(x^{jq}, 1) \log \frac{1}{P_{lj}^{pq}(s)}]$
$d = \sum_{j \neq l} \sum_{q=1}^{k_j} [n(x^{jq}, 0) \log \frac{1}{1 - P_{lj}^{pq}(s)}]$
20: **end for**
21: ### terminate the whole program
22: **if** there is no decrease in Equation (3.2) **then**
23: Break.
24: **else**
25: **for** $l = 1, \ldots, m$ **do**
26: Set $\Phi_l(s+1) = \Phi_l'(s)$.
27: **end for**
28: Set $s \leftarrow s + 1$.
29: **end if**
30: **end while**

linearly, i.e., to increase the cluster number by 1 instead of doubling it. In order to decide which cluster to split, we use the procedure in Alg. 3, which is based on maximum entropy criterion.

The correctness of **PaCK** is given by theorem 3.2

THEOREM 3.2. *(1) The outer loop of* **PaCK** *(i.e. step 6) never increases the objective function (Equation* (2.1)*); (2)* **PaCK** *converges in finite steps.*

Proof. Omitted for brevity.■

Note that **PaCK** stops when the total coding cost (Equation (2.1)) does not decrease. Therefore, **PaCK** can be seen as a model selection procedure. Given a specific model (the number of clusters for each object type), we find the parameters (the mappings from object to object clusters) according to the empirical risk (Equation (3.2)). Then we evaluate different models based on the regularized empirical risk (Equation (2.1)). In this way, **PaCK** avoids over-fitting, i.e., generating a large number of clusters where each object corresponds to an individual cluster.

Although we assume that there is connectivity between each pair of object types, **PaCK** can be easily generalized to the cases where some object types are not connected with each other. In this case, the corresponding terms in the objective function (Equation (3.2)) and the total coding cost (Equation (2.1)) disappear.

3.3 Analysis of Computational Complexity.

In this subsection, we analyze the computational complexity of the proposed algorithms.

First, in each iteration of the proposed **CCsearch** algorithm, for object type $\mathcal{X}_i$, we need to count the number of non-zero elements in each row, and assign it to one of the k_i clusters. Therefore, we have the following lemma for the proposed **CCsearch** algorithm:

LEMMA 3.1. *The computational complexity for each iteration of* **CCsearch** *algorithm is* $O(k_i \cdot \sum_{j \neq i} n(D_{ij}, 1))$. *Let* I *denote the total number of iterations. The overall complexity for* **CCsearch** *algorithm is* $\sum_{i=1}^{m}(k_i \cdot \sum_{j \neq i} n(D_{ij}, 1)) \cdot I$.

Proof. Omitted for brevity.■

Next, we analyze the complexity of the **PaCK** algorithm. Let k^*_{max} be the total number of times that **CCsearch** is called in **PaCK** and let I_{max} be the maximum number of iteration steps each time we run the **CCsearch** algorithm. We have the following lemma for the proposed **PaCK**:

LEMMA 3.2. *The overall complexity of the* **PaCK** *algorithm is* $O(k^*_{max} I_{max} \cdot \sum_{i=1}^{m-1}(k_i^* \cdot \sum_{j \neq i} n(D_{ij}, 1)))$.

Proof. Omitted for brevity.■

Finally, notice that in each iteration of the outer loop of **PaCK** (step 6), we will call **CCsearch** at most $2m$ times (i.e., at most twice for each type of object). And also,

Algorithm 2 PaCK

Input: The connectivity matrices $D_{i,j}$, $(i, j = 1, ..., m)$.
Output: k_i^* and the corresponding mapping Φ_i^S for $i = 1, \ldots, m$.

1: Let S denote the search iteration index.
2: Initialize $S = 0$ and $k_i = 1, i = 1, \ldots, m$.
3: **for** $l = 1, ..., m$ **do**
4: Initialize type l as 'split successfully'.
5: **end for**
6: **while** true **do**
7: ### alternate among different type of objects
8: **for** $l = 1 : m$ **do**
9: ### try to update the cluster number for type l
10: **if** type l is marked as 'split successfully' **then**
11: ### try exponential split
12: Set $k_l^{S+1} = 2 * k_l^S$.
13: **else**
14: ### try linear split
15: Set $k_l^{S+1} = k_l^S + 1$.
16: **end if**
17: Concatenate all the matrices D_{lj}, $j \neq l$, to form a single matrix D_l, which is an $n_l \times \sum_{j \neq l} n_j$ matrix.
18: Construct an initial mapping $\Phi_l^{'S+1}$ using **InitialSearch**.
19: Use **CCsearch** to find new mappings $\Phi_1^{S+1}, \ldots, \Phi_m^{S+1}$.
20: **if** there is no decrease in Equation (2.1) **then**
21: Set $k_l^{S+1} = k_l^S$, $\Phi_1^{S+1} = \Phi_1^S, \ldots, \Phi_m^{S+1} = \Phi_m^S$.
22: Mark type l as 'split un-successfully'.
23: **else**
24: Mark type l as 'split successfully'.
25: **end if**
26: ### try to merge two clusters
27: **if** $\sum_{i=1}^m k_i^{S+1} > S + 1$ **then**
28: Randomly select two clusters of type l.
29: Merge the two selected clusters.
30: Use **CCsearch** to find new mappings $\tilde{\Phi}_1^{S+1}, \ldots, \tilde{\Phi}_m^{S+1}$.
31: **if** $\tilde{\Phi}_1^{S+1}, \ldots, \tilde{\Phi}_m^{S+1}$ produce decrease in Equation (2.1) **then**
32: ### successful merge
33: Set $\Phi_1^{S+1} = \tilde{\Phi}_1^{S+1}, \ldots, \Phi_m^{S+1} = \tilde{\Phi}_m^{S+1}$.
34: Update $k_l^{S+1} \leftarrow k_l^{S+1} - 1$.
35: **end if**
36: **end if**
37: **end for**
38: Update $S \leftarrow S + 1$.
39: ### terminate the whole program
40: **if** $\sum_{l=1}^m k_l^{S+1} = \sum_{l=1}^m k_l^S$ **then**
41: Set $k_l^* = k_l^S, l = 1, \ldots, m$.
42: Set $S^* = S$.
43: Break.
44: **end if**
45: **end while**

Algorithm 3 InitialSearch

Input: The connectivity matrices D_{lj}, $j \neq l$, and the original mapping Φ_l with k_l clusters.

Output: Initial mapping Φ_l' with $k_l + 1$ clusters.

1: Split the row group r with the maximum entropy per row, i.e.

$$r := \arg \max_{1 \leq p \leq k_l} \sum_{j \neq l} \sum_{q=1}^{k_j} \frac{n(D_{lj}^{pq}) H(P_{lj}^{pq})}{a_l^p}$$

2: Construct Φ_l' as follows. For every row x in row group r, place it into the new group $k_l + 1$ if and only if it decreases the per-row entropy of group r, i.e.,

$$\sum_{j \neq l} \sum_{q=1}^{k_j} \frac{n(\tilde{D}_{lj}^{rq}) H(\tilde{P}_{lj}^{rq})}{a_l^r - 1} < \sum_{j \neq l} \sum_{q=1}^{k_j} \frac{n(D_{lj}^{rq}) H(P_{lj}^{rq})}{a_l^r}$$

where $\tilde{D}_{lj}^{rq}$ is D_{lj}^{rq} without row x, and $\tilde{P}_{lj}^{rq}$ is the bias of $\tilde{D}_{lj}^{rq}$. Otherwise, we place x in the original group. If we move the row to the new group, we update D_{lj}^{rq} and P_{lj}^{rq}, $j \neq l$, $q = 1, \ldots, k_j$ by removing row x.

we only invoke the merge operation when the condition $\sum_{l=1}^{m} k_l^{S+1} > S + 1$ holds (step 27 in **PaCK**, where S is the iteration index for the outer loop of **PaCK**). Therefore, we have the following lemma for the total number of times that **CCsearch** is called in **PaCK**.

LEMMA 3.3. *k_{max}^* in lemma 3.2 is upper bounded by $2m \sum_{l=1}^{m} k_l^*$.*

Proof. Omitted for brevity.■

4 Experimental Results

In this Section, we evaluate the performance of **PaCK**. The experiments are designed to answer the following two questions:

- How good is the search quality of **PaCK**?
- How fast is **PaCK**?

4.1 Experimental Setup

Data sets. We use both synthetic and real data sets. For synthetic data sets, given the number of object types, we first specify the connectivity pattern among the different types, such as line-shape, star-shape, loop-shape and so on. Figure 1 illustrates the connectivity patterns used in our experiments. Then within each object type, we generate clusters of different sizes. Finally, we randomly flip the elements of the connectivity matrices with a certain probability (i.e., the noise level).

In addition to the synthetic data sets, we also test **PaCK** on two real data sets: the 20 newsgroups data set[3] and the NIPS data set[4].

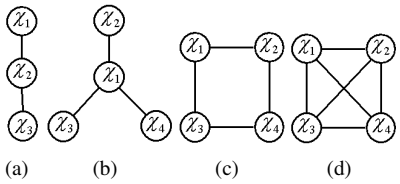


Figure 1: Different connectivity patterns: (a) line; (b) star; (c) loop; (d) clique.

For 20 newsgroups data set, we use the documents from 4 different domains ('talk', 'sci', 'rec' and 'comp') to form a star shape k-partite ($k = 5$) graph, where the 'documents' from each domain is treated as one type of objects at the leaf and the 'words' as another type of objects in the center. The connectivity structure of this data set is shown in Figure 2. Altogether, there are 61,188 words, 16,015 documents and 4,140,814 edges.

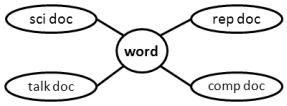


Figure 2: The connectivity structure for 20 newsgroups data.

For the NIPS data set, we use the papers from 1987 to 1999 NIPS proceedings to construct a 'keyword-paper-author' tri-partite graph, with 'paper' in the middle. The 'keyword' is extracted from the paper titles by removing the stop words (such as 'a', 'the', 'and', etc). The connectivity structure of this data set is shown in Figure 3. Altogether, there are 2,037 authors, 1,740 papers, 2,151 keywords and 458,995 edges.

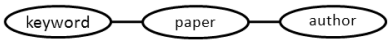


Figure 3: The connectivity structure for NIPS data.

Comparison methods. To the best of our knowledge, there are no parameter-free methods for clustering k-partite graphs in the literature. For comparison, we have designed the following type-oblivious method based on Autopart [6] as the baseline. In the type-oblivious method, we first form a big connectivity matrix, where the rows and columns correspond to all the objects, and then apply Autopart based on this big matrix. Note that in this way, the generated clusters may consist of objects of different types. To address this problem, we add a post-processing step to further decompose such heterogeneous clusters into homogeneous clusters, i.e., clusters of the same type of objects. We also compare the proposed **PaCK** with the two most recent clustering methods for k-partite graphs: the collective matrix factorization method [20, 21] (referred to as 'CollMat') and the spectral relational clustering algorithm [16] (referred to as 'Spec').

[3] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html

[4] http://www.cs.toronto.edu/~roweis/data.html

Both 'CollMat' and 'Spec' require the user to provide the cluster number as inputs.

Evaluation metric. To evaluate the quality of the clusters generated by **PaCK**, we adopt the measurement from [1]. Specifically, given two clusterings $B = B_1 \cup \ldots \cup B_k$ and $B' = B'_1 \cup \ldots \cup B'_{k'}$ that partition the objects into k and k' subsets, define $d^2(B, B') = k + k' - \sum_{i,i'} \frac{|B_i \cap B'_{i'}|^2}{|B_i| \times |B'_{i'}|}$, where $B_i \cap B'_{i'}$ is the common objects in B_i and $B'_{i'}$, and $|\cdot|$ denotes the set cardinality. Note that in its original form, $d^2(B, B')$ is between 0 (if B and B' are exactly the same) and $k + k' - 2$ (if $k' = 1$). In our application, we normalize the distance so that it is always between 0 and 1, i.e., $d^2(B, B') = (k + k' - \sum_{i,i'} \frac{|B_i \cap B'_{i'}|^2}{|B_i| \times |B'_{i'}|})/(k + k' - 2)$. For each object type $\mathcal{X}_i$, we compare the clustering generated by **PaCK** and the ground truth using this distance function to get d_i^2. Then we calculate the average distance over all the object types, i.e., $\bar{d}^2 = \sum_{i=1}^{m} d_i^2/m$. Based on the above discussion, smaller values of $\bar{d}^2$ indicate better clustering for all object types.

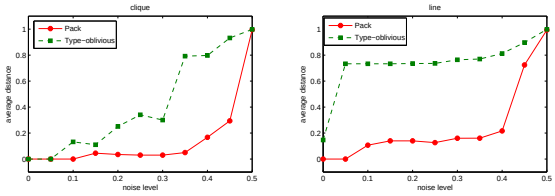
Machine configurations. All running time reported in this paper is measured on the same machine with four 3.0GHz Intel Xeon CPUs and 16GB memory, running Linux (2.6 kernel). Unless otherwise stated, all the experimental results are averaged over 20 runs.

4.2 Quality Assessment on Synthetic Data Sets.

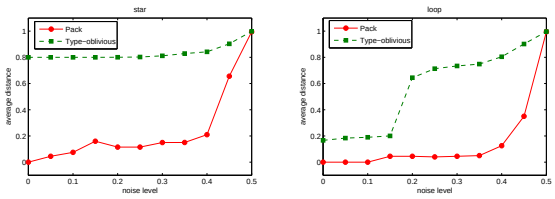
Figure 4 compares the clustering quality of **PaCK** and the type-oblivious method on four connectivity patterns. Both methods are parameter-free. From these figures, we can make the following conclusions: (1) in all the cases, if there is no noise, **PaCK** is able to recover the exact clusterings for different types of objects simultaneously (i.e., average distance is 0); (2) the quality of **PaCK** is quite robust against the noise level; (3) in terms of the clustering quality, **PaCK** is always significantly better than the type-oblivious method.

We also compare **PaCK** with 'CollMat' [20, 21] and 'Spec' [16]. Since both 'CollMat' and 'Spec' require the cluster numbers as inputs, we tune the true cluster numbers for all the three methods for a fair comparison. Note that in this case, **PaCK** degenerates to **CCsearch**. Figure 5 presents the comparison results. In most cases, our **PaCK** consistently outperforms both 'CollMat' and 'Spce' .

Figure 6 illustrates one run of **PaCK** on loop-shape connectivity, with noise level 20%. In this figure, the first sub-figure shows a set of original connectivity matrices. In the second sub-figure, the rows and columns of each connectivity matrix are rearranged so that the objects from the same cluster are put together. A dark block means that it has a large proportion of 1s, and a white block means that it has a small proportion of 1s. From this figure, we see that the clusters generated by **PaCK** are quite homogeneous: either

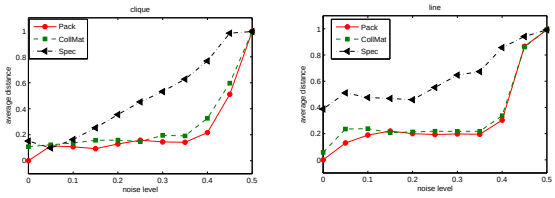


(a) Clique-shape (b) Line-shape

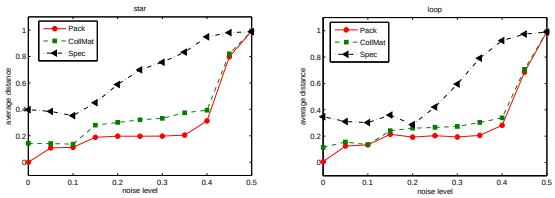


(c) Star-shape (d) Loop-shape

Figure 4: Comparison of **PaCK** vs. type-oblivious on synthetic data sets. (Smaller is better.)



(a) Clique-shape (b) Line-shape

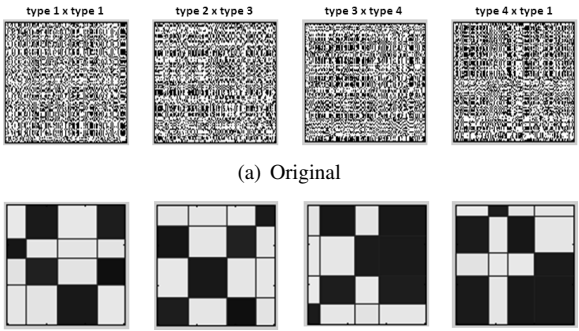


(c) Star-shape (d) Loop-shape

Figure 5: Comparison of **PaCK** vs. 'CollMat' and 'Spce' on synthetic data sets. (Smaller is better.)

quite dark or quite white.

If we ignore the exponential split, merge operation and multiple concurrent trials, the proposed **PaCK** (referred as '**PaCK**-Basic') is similar to the outer loop of cross-associations except that in '**PaCK**-Basic', we will try to alternate among m, instead of 2, types of objects. We evaluate the benefit of those additional operations (i.e., introducing exponential split, merge as well as multiple concurrent trials). The results are presented in Figure 7. For the multiple concurrent trials part, we run 10 trials of the proposed **PaCK** and pick up the one which gives the lowest coding cost. In our case, we use parallelism (i.e., multiple concurrent trials) to improve the search quality, instead of search speed. The average distance is normalized by the **PaCK**-Basic. From Figure 7, we can see that these additional operations (i.e., introducing exponential split, merge and multiple concurrent

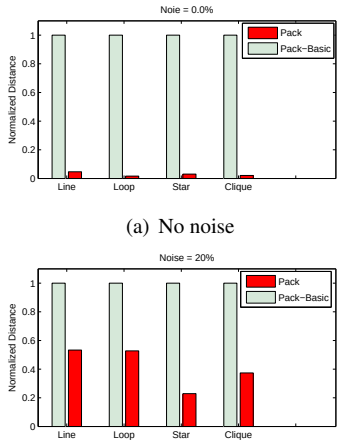


(a) Original

(b) After re-ordering

Figure 6: Clustering results of **PaCK** on loop-shape connectivity.

trials) largely improve search quality: in all the cases, the average distance of **PaCK** is only a fraction of that by **PaCK**-Basic.



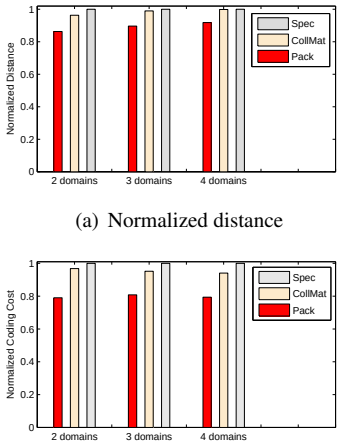
(a) No noise

(b) 20% noise

Figure 7: Comparison of search procedures. (Smaller is better.)

4.3 Quality Assessment on 20 Newsgroups Data Set.

We use the 20 newsgroups data set to compare the proposed **PaCK** with the two most recent clustering algorithms for k-partite graphs: the collective matrix factorization method [20, 21] (referred to as 'CollMat') and the spectral relational clustering algorithm [16] (referred to as 'Spec'). Both 'CollMat' and 'Spec' require the user to provide the cluster numbers as inputs. Therefore, we use the true cluster number for all the three methods ('CollMat', 'Spec' and '**PaCK**') for a fair comparison. The overall graph for this data set is a 5-partite graph with 'word' object in the center. We also randomly select a subset from all four 'document' objects and form a smaller star-shape k-partite ($2 \leq k \leq 5$) graphs. The results are presented in Figure 8(a). Since we do not have the ground truth for the 'word' object, the cluster distance is averaged over all 'document' objects in the corresponding graph and it is normalized by the highest values among three different methods. It can be seen that

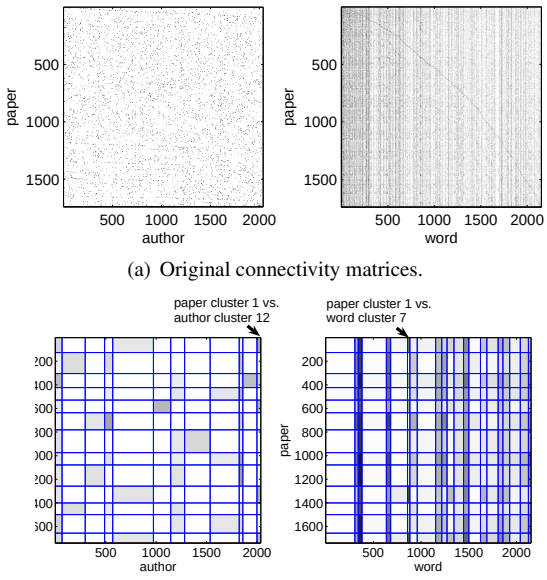


(a) Normalized distance

(b) Normalized coding cost

Figure 8: Comparison of clustering results for 20 newsgroups data. (Smaller is better.)

in all cases, the proposed **PaCK** performs the best. In Figure 8(b), we also present the final normalized coding cost for the three methods. It is interesting to notice that **PaCK** also achieves the lowest coding cost, which indicates that there might be a close relationship between the coding cost and the clustering quality. Finally, it is worth pointing out that both 'CollMat' and 'Spec' do not work if the cluster numbers are not given by the users. On the other hand, **PaCK** will try to search for such parameters automatically.



(a) Original connectivity matrices.

(b) Connectivity matrices after reordering.

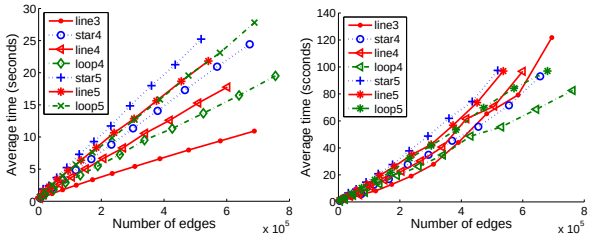
Figure 9: Clustering result on NIPS data.

4.4 Quality Assessment on NIPS Data Set.

Unlike the synthetic data sets and the 20 newsgroups data set, for the NIPS data set, we do not have the ground

Some papers: from `Paper' cluster 1
Information Maximization in Single Neurons
Channel Noise in Excitable Neural Membranes
Optimal Neural Spike Classification
Simulation and Measurement of the Electric Fields Generated by Weakly Electric Fish
Cholinergic Modulation May Enhance Cortical Associative Memory Function
Non-Boltzmann Dynamics in Networks of Spiking Neurons
Optimal Filtering in the Salamander Retina
Distributed Synchrony of Spiking Neurons in a Hebbian Ceil Assembly
Signal Detection in Noisy Weakly-Active Dendrites
Some authors from `Author' cluster 12
Bower_J; Baird_B; Bialek_W; Deco_G; Goodhill_G; Horn_D; Koch_C; Maass_W; Meilijson_I; Pouget_A; Ruppin_E; Schulten_K; vanHemmen_J
Sample words `Key Word' cluster 7
cortex; neurons; synaptic; activity; brain; spatial; neuron; mechanism; biological; stimulus; cells; response; frequency; activity.

Figure 10: An example of the resulting 'author-paper-keyword' clusters.



(a) 0 noise (b) 10% noise

Figure 11: Wall-clock time versus the total number of edges. The number following each connectivity pattern is the number of object types.

truth. Therefore, we use this data set as a case study to illustrate the effectiveness of **PaCK**. The original connectivity matrices ('paper' versus 'author', and 'paper' versus 'keyword') are shown in Figure 9(a). Using **PaCK**, we find 13 'paper' clusters, 12 'author' clusters and 22 'keyword' clusters. Figure 9(b) plots the connectivity matrices after rearranging based on the clustering results. Using these reordered matrices, we have a concise summary of the original data set, e.g., we can see that 'author group' 12 is working on the same topic ('paper' group 1), using the same set of keywords ('keyword' group 7). We manually verify that the topic is on 'neural information process', and report in Figure 10 some sample papers, authors and keywords from each of these clusters. Other clusters found by **PaCK** are also consistent with human intuition, e.g., 'paper' cluster 2 is about statistical machine learning; 'paper' cluster 12 is about computer vision and so on.

4.5 Evaluation of Speed.

According to our analysis in Subsection 3.3, the complexity of **PaCK** grows linearly with the total number of edges in the connectivity matrices. Figure 11 shows the wall-clock time of **PaCK** versus the total number of edges in the connectivity matrices with different noise levels. This figure illustrates that when there is no noise (the left sub-figure), the curves are straight lines. When the noise level is 10% (the right sub-figure), the curves are close to straight lines, and the deviations may be due to the different number of iteration steps in the **CCsearch** algorithm.

5 Related Work

The idea of using compression for clustering can be traced back to the information-theoretic co-clustering algorithm [8], where the normalized non-negative contingency table is treated as a joint probability distribution between two discrete random variables that take values over the rows and columns. The optimal co-clustering is the one that minimizes the difference in mutual information between the original random variables and the mutual information between the clustered random variables.

As mentioned in Section 1, the information-theoretic co-clustering algorithm can only be applied to bipartite graphs. However, the idea behind this algorithm can be generalized to more than two types of heterogeneous objects. For example, in [10], the authors proposed the CBGC algorithm. It aims to do collective clustering for star-shaped inter-relationships among different types of objects. Followed up work includes the high order co-clustering [12]. Another example is the spectral relational clustering algorithm proposed in [16]. Unlike the previous algorithm, this algorithm is not restricted to star-shaped structures. More recently, the collective matrix factorization proposed by Singh *et al.* [20, 21] can also be used for clustering k-partite graphs.

Despite of their success, one major drawback of the above algorithms is that they all require the user to specify certain parameters. In the information-theoretic co-clustering algorithm, the user needs to specify the numbers of row clusters and column clusters. In both the CBGC algorithm and the spectral relational clustering algorithm, besides giving the number of clusters for each type of objects, the user also needs to specify reasonable weights for different types of relations or features. However, in real applications, it might be very difficult to determine the number of clusters for clustering algorithms, especially when the data set is very large, not to mention the challenge of specifying the weights. On the other hand, the proposed **PaCK** is totally parameter-free, i.e., it requires no user intervention.

In terms of parameter-free clustering algorithms for graphs, cross-associations in [7], which is designed for bipartite graph, is most representative. Similarly, the algorithm (Autopart) proposed in [6] also tries to find the number of clusters and the corresponding clustering for a unipartite graph in a parameter-free fashion. Both cross-associations [7] and Autopart [6] do not apply to k-partite graphs when $k > 2$. The proposed **PaCK** generalizes the idea of cross association/autopart so that it can deal with multiple types of objects. In terms of problem formulation, **PaCK** is similar to cross-associations/Autopart, except that in **PaCK**, we try to compress multiple, instead of a single, matrices collectively. In fact, if we ignore the

type information and treat the whole heterogeneous graph as one big (unipartite) graph, conceptually, it seems that we can directly leverage Autopart for clustering. However, as we show in the experimental section, this strategy ('type-oblivious') usually leads to poor performance exactly because the type information is ignored. Moreover, we carefully design the search procedure (e.g., by introducing exponential split, merge, multiple concurrent trials, etc) in the proposed **PaCK**, which largely improve the search quality as we show in the experimental section. On the other hand, the proposed **PaCK** inherits the two important merits from the original cross association/autopart: (1) *parameter-free* and (2) *scalability*, which are very important for many real applications.

Other related work includes (1) GraphScope [23], which uses a similar information-theoretic criterion as cross association for time-evolving graphs to segment time into homogeneous intervals; and (2) multi-way distributional clustering (MDC) [4] which is demonstrated to outperform the previous information-theoretic clustering algorithms by the time the algorithm was proposed. However, in MDC, we still need to tune the weights for different connectivity matrices and it is not clear what its computational complexity is in big-O notations. On the other hand, our **PaCK** is parameter-free and it is clearly linear on the number of the edges in the graph.

6 Conclusion

In this paper, we have proposed **PaCK** to cluster k-partite graphs, which to the best of our knowledge, is the first parameter-free method to cluster k-partite graphs. In terms of problem formulation, **PaCK** seeks a good compression for multiple matrices collectively. We carefully design the search procedure in **PaCK** so that (1) it can find a good approximate solution, and (2) the whole algorithm is scalable in the sense that it is linear on the number of edges in the graphs. The major advantage of **PaCK** over all existing methods for clustering k-partite graphs (CBGC algorithm, the spectral relational clustering algorithm and collective matrix, etc.) lies in its parameter-free nature. Furthermore, **PaCK** can be easily generalized to the cases where certain connectivity relations form tensors. We verify the effectiveness and efficiency of **PaCK** by extensive experimental results.

Acknowledgement. Jingrui He was supported in part by IBM Research PhD Fellowship.

References

[1] F. Bach and Z. Harchaoui. Diffrac: a discriminative and flexible framework for clustering. In *NIPS*, 2007.

[2] F. Bach and M. Jordan. Learning spectral clustering. In *NIPS*, 2003.

[3] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. *The Journal of Machine Learning Research*, 8:1919–1986, October 2007.

[4] R. Bekkerman, R. El-Yaniv, and A. McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML*, pages 41–48, New York, NY, USA, 2005. ACM.

[5] T. Berners-Lee. Giant Global Graph.

[6] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004.

[7] D. Chakrabarti, S. Papadimitriou, D. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88, 2004.

[8] I. Dhillon, S. Mallela, and D. Modha. Information-theoretic co-clustering. In *KDD*, pages 89–98, 2003.

[9] R. Duda, P. Hart, and D. Stork. Pattern classification. 2001.

[10] B. Gao, T. Liu, X. Zheng, Q. Cheng, and W. Ma. Consistent bipartite graph co-partitioning for star-structured high-order heterogeneous data co-clustering. In *KDD*, pages 41–50, 2007.

[11] E. Gokcay and J. Principe. Information theoretic clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:158–171, February 2002.

[12] G. Greco, A. Guzzo, and L. Pontieri. An information-theoretic framework for high-order co-clustering of heterogeneous objects. In *SEBD*, pages 397–404, 2007.

[13] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[14] D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *VLDB*, pages 13–23. VLDB Endowment, 2004.

[15] T. Li. A general model for clustering binary data. In *KDD*, pages 188–197, 2005.

[16] B. Long, Z. Zhang, X. Wu, and P. Yu. Spectral clustering for multi-type relational data. In *ICML*, pages 585–592, 2006.

[17] B. Long, Z. Zhang, and P. Yu. A probabilistic framework for relational clustering. In *KDD*, pages 470–479, 2007.

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

[19] T. Mitchell. *Machine Learning*. McGraw-Hill Science Engineering, 1997.

[20] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.

[21] A. P. Singh and G. J. Gordon. A unified view of matrix factorization models. In *ECML/PKDD (2)*, pages 358–373, 2008.

[22] N. Slonim, N. Friedman, and N. Tishby. Unsupervised document classification using sequential information maximization. In *SIGIR*, pages 129–136, New York, NY, USA, 2002. ACM.

[23] J. Sun, C. Faloutsos, S. Papadimitriou, and P. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696, 2007.

[24] W3C. Resource Description Framework.