Fast Information-Theoretic Agglomerative Co-clustering

Tiantian Gao and Leman Akoglu

Stony Brook University
Department of Computer Science
{tiagao,leman}@cs.stonybrook.edu

Abstract. Jointly clustering the rows and the columns of large matrices, a.k.a. co-clustering, finds numerous applications in the real world such as collaborative filtering, market-basket and micro-array data analysis, graph clustering, etc. In this paper, we formulate an information-theoretic objective cost function to solve this problem, and develop a fast *agglomerative* algorithm to optimize this objective. Our algorithm rapidly finds highly similar clusters to be merged in an iterative fashion using Locality-Sensitive Hashing. Thanks to its bottom-up nature, it also enables the analysis of the cluster hierarchies. Finally, the number of row and column clusters are automatically determined without requiring the user to choose them. Our experiments on both real and synthetic datasets show that the proposed algorithm achieves high-quality clustering solutions and scales linearly with the input matrix size.

1 Introduction

Clustering is a widely used technique that aims to group similar objects together, with numerous applications such as data summarization, classification, and outlier detection. Typically, the input data is represented as a two-mode matrix, e.g. customer-product purchasing data, document-term occurrence data, user-webpage browsing data, etc. Traditional clustering focuses only on one-mode, that is, clustering one dimension of the data matrix based on similarities along the second dimension, e.g., document clustering based on term similarity.

Another class of methods focuses on the *two-mode* clustering problem (a.k.a. bi-, co-, or block clustering), which aims at *simultaneously* clustering both dimensions of the data matrix, e.g. document clusters based on term similarity together with term clusters based on document appearance similarity. An illustration of co-clustering is given in Fig. 1. Co-clustering has many applications such as micro-array data analysis, market-basket analysis, (bi-partite) graph clustering, to name but a few. The main advantage of co-clustering is that the joint clustering of the rows and columns fully and succinctly summarizes the underlying structure of relations in the data for both types of objects.

In this paper, we propose a fast agglomerative hierarchical co-clustering technique, that scales linearly with the input matrix size. Our motivation is that agglomerative clustering techniques are known to alleviate the resolution-limit problem in clustering [10], being able to find smaller size clusters effectively. Our proposed algorithm, called CoClusLSH, rapidly finds the most similar objects

H. Wang and M.A. Sharaf (Eds.): ADC 2014, LNCS 8506, pp. 147–159, 2014.
© Springer International Publishing Switzerland 2014

to merge, using ideas from Locality-Sensitive Hashing, and iteratively builds the row and column cluster hierarchies. The two hierarchies are built in an alternating fashion, such that the clustering of both object types is intertwined.

In clustering, one of the main challenges is to determine the "correct" or a "good" number of clusters. In hierarchical clustering, this challenge translates to picking a level of the hierarchy to "cut", the subtrees of which determine the final clustering. It is often a hard task for the user to specify the number of output clusters, especially for large datasets. We circumvent this challenge by formulating an information-theoretic co-clustering objective cost function, based on the number of bits needed to encode the input matrix. Our goal then is to find a clustering that achieves as low of a cost as possible. We update this cost while growing up our cluster hierarchies and merge wo clusters only when it yields a lower cost. This principled way of building the clustering is exactly what guides us in "when to stop"—stop growing the hierarchies when no further merges can reduce the objective cost. As such, the number of sub-hierarchies at algorithm termination automatically gives us the number of row and column clusters.

The main merits of our method over the (cited) previous proposals are that (*i*) it automatically finds a good number of clusters [9,19], (*ii*) achieves linear scalability [12], and (*iii*) provides the cluster hierarchies [7] (see §4 for details). None of the previous approaches exhibits all three properties at the same time. We summarize our main contributions as follows:

- We propose a new technique for agglomerative co-clustering, and formulate an information-theoretic objective that enables us to determine the number of row/column clusters automatically in a principled data-driven way,
- We develop a fast algorithm called CoClusLSH that rapidly finds similar clusters to merge in order to grow the row/column hierarchies,
- We show that CoClusLSH scales linearly with the input matrix size,
- Experiments on synthetic and real datasets with ground truth cluster labels demonstrate the effectiveness and efficiency of our method.

2 Proposed Method

2.1 Problem Definition

We consider the problem of *co-clustering*, i.e. joint clustering the rows and columns of a large binary matrix (such a matrix can be thought of as a bipartite graph). In particular, given a bipartite graph with n type-1 nodes, m type-2 nodes, and their binary connectivity information, our goal is to cluster the type-1 nodes into k, and the type-2 nodes into l disjoint clusters such that the nodes in

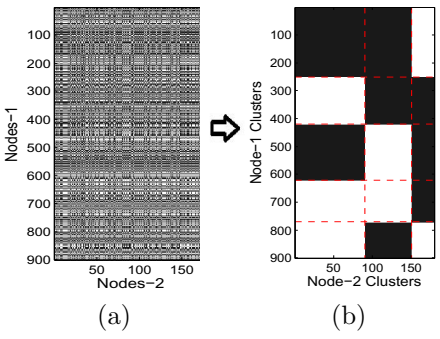

Fig. 1. (a) Example graph with n=900, and m=180, where (b) CoClusLSH finds k=5 type-1 and l=3 type-2 clusters.

the same cluster have "similar connectivity". Intuitively, a set of nodes have similar connectivity if their neighbors "highly" overlap (for e.g., see Fig. 1).

Given the above problem description, two main questions arise: (P1) how to choose the number of node clusters k and l?, and (P2) how to assign the nodes to their "proper" clusters? (P2) aims at summarizing the adjacency matrix $\mathbf{A}$ of the graph with homogeneous, rectangular regions of high and low densities, while (P1) deals with choosing the right number of clusters and hence the number of these rectangular regions. Roughly speaking, having more clusters allows us to obtain more homogeneous regions. At the very extreme we can have $n \times m$ "regions" each with perfect 0 or 1 density which, however, does not provide any summary. As such, a co-clustering algorithm should achieve a good trade-off between homogeneity and the number of regions. Intuitively, this trade-off calls for model selection, which brings us to the next section.

2.2 Problem Formulation

In order to achieve a proper balance between the homogeneity and the number of the rectangular regions, we use a similar objective function to [7] founded on the Minimum Description Length (MDL) principle [25]. MDL provides a model selection criterion based on lossless compression principles, where the objective is to compress/transmit/store the adjacency matrix $\mathbf{A}$ using as few bits as possible. The compression cost consists of two main parts: the number of bits required to encode (1) the clustering "summary" (model description cost), and (2) each rectangular region (data description cost) given the model.

Next we describe each part in detail in the context of our objective function after providing the notation.

Notation. Let k and l respectively denote the number of disjoint row- and column-clusters, and $R : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, k\}$ and $C : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, l\}$ denote the assignments of rows to row-clusters and columns to column-clusters. We refer to (R, C) as a *mapping*. To better describe a mapping, let us rearrange the rows and columns of the adjacency matrix $\mathbf{A}$ such that all rows corresponding to row-cluster-1 are listed first, followed by rows in row-cluster-2, and so on. We also rearrange the columns in a similar fashion using column-cluster assignments. One can imagine that such a rearrangement sub-divides $\mathbf{A}$ into $k \times l$ two-dimensional, rectangular blocks (as in Fig. 1 (b)), which we will refer to as B_{ij}, $i = 1, \ldots, k$ and $j = 1, \ldots, l$. Finally, let (r_i, c_j) denote the dimensions of B_{ij}, where r_i denotes the size of row cluster i, and c_j denotes the size of column cluster j.

Objective Function. Our objective function consists of a two-part (lossless) compression cost of the adjacency matrix $\mathbf{A}$. This cost can be thought of as the total number of bits required to encode $\mathbf{A}$. The first part is the model description cost that consists of describing the mapping (R, C). The second part is the data description cost that consists of encoding the sub-matrices (i.e., the B_{ij} "blocks"), given the mapping. Intuitively, a good choice of (R, C) would compress $\mathbf{A}$ well, and yield a low total description cost. In particular:

The *Model Description Cost* consists of encoding the number of row and column clusters and the corresponding mapping.

- The matrix dimensions of $\mathbf{A}$ require $\log^\star n + \log^\star m$ bits, where $\log^\star$ denotes the universal code length for integers.[1] This term is independent of any particular mapping.
- The number of row and column clusters (k, l) require $\log^\star k + \log^\star l$ bits.
- The row and column cluster assignments with arithmetic coding require $nH(P) + mH(Q)$ bits, where H denotes the Shannon entropy function, P is a multinomial random variable with the probability $p_i = \frac{r_i}{n}$ and r_i is the size of the i-th row cluster, $1 \leq i \leq k$. Similarly, Q is another multinomial random variable with the probability $q_j = \frac{c_j}{m}$ and c_j is the size of the j-th column cluster, $1 \leq j \leq l$.

The *Data Description Cost* consists of encoding the actual blocks.

- For each block B_{ij}, $i = 1, \ldots, k$, $j = 1, \ldots, l$, encoding $n_1(B_{ij})$, i.e. the number of 1s it contains, takes $\log_2(r_i c_j + 1)$ bits.
- To encode the actual blocks B_{ij}, we first calculate their density $P_{ij}(1) = n_1(B_{ij})/n(B_{ij})$, where $n(B_{ij}) = n_1(B_{ij}) + n_0(B_{ij}) = r_i c_j$. Then, the number of bits required to encode each block can be written as:

$$E(B_{ij}) = -n_1(B_{ij}) \log_2(P_{ij}(1)) - n_0(B_{ij}) \log_2(P_{ij}(0)) = n(B_{ij}) H(P_{ij}(1)).$$

Overall, the *Total Encoding Cost (Length $L(\mathbf{A}; R, C)$ in bits)* becomes

$$L(\mathbf{A};R,C) = \log^\star n + \log^\star m + \log^\star k + \log^\star l + \sum_{i=1}^{k} r_i \log_2(\frac{n}{r_i}) + \sum_{j=1}^{l} c_j \log_2(\frac{m}{c_j}) + \sum_{i=1}^{k}\sum_{j=1}^{l} \Big(\log_2(r_i c_j + 1) + E(B_{ij}) \Big) \quad (1)$$

2.3 Proposed Algorithm CoClusLSH

Minimizing our objective function in Equ. (1) is intractable for very large graphs as the number of possible orderings of rows/columns is combinatorial. Thus, we develop an algorithm that aims at finding a fast approximate solution.

Our CoClusLSH algorithm starts by assigning each row and column in $\mathbf{A}$ to their respective clusters. In the main loop, it alternates between trying to merge candidate column and row clusters, for reduced cost. In order to rapidly find sufficiently similar candidate clusters, it employs the LSH technique [11] and generates a *signature* for each cluster which is then used to hash the clusters into multiple hash tables. Candidate clusters hashed to the same buckets are then tested for merge. The algorithm terminates when no more merges can be done for lower cost. The clusters at termination constitutes the final set of clusters, and the intermediate merge operations define the cluster hierarchies.

We provide the detailed pseudocode for CoClusLSH in Algorithm 1.

[1] The optimal number of bits required to encode a positive integer x whose range is unknown is $\log^\star x \approx \log_2 x + \log_2 \log_2 x + \ldots$ of the positive terms [25].

Algorithm 1. CoClusLSH

Input: $n \times m$ adjacency matrix $\mathbf{A}$, LSH parameters r, b
Output: A heuristic solution towards minimizing total encoding $L(\mathbf{A}; R, C)$: number of row and column groups (k^*, l^*), associated mapping (R^*, C^*)

1. Set R^0:=$\{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ Set C^0:=$\{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$
2. Set k^0=n, l^0=m. Let T denote the outer iteration index. Set $T = 0$.
3. **repeat**
4. C^{T+1}, l^{T+1}:=Merge-ColClus($\mathbf{A}$, C^T, l^T, k^T, r, b)
5. R^{T+1}, k^{T+1}:=Merge-RowClus($\mathbf{A}$, R^T, k^T, l^T, r, b)
6. **if** $L(\mathbf{A}; R^{T+1}, C^{T+1}) \geq L(\mathbf{A}; R^T, C^T)$ **then**
7. **return** (k^*, l^*)=(k^T, l^T), (R^*, C^*)=(R^T, C^T)
8. **else** Set $T = T + 1$ **end if**
9. **until** convergence

Procedure 1. Merge-ColClus (**Procedure 2** Merge-RowClus is similar)

Input: $n \times m$ adjacency matrix $\mathbf{A}$, C^T, l^T, k^T, LSH parameters r, b
Output: C^{T+1}, l^{T+1}

1. {**Step 1. Generate signatures**} initialize signature matrix $S[i][j] \in \mathbb{R}^{rb \times l^T}$
2. **if** $T = 0$ **then** {use Jaccard similarity *// generate min-hash signatures* }
3. **for** $i = 1$ to rb **do**
4. $\pi_i \leftarrow$ generate random permutation $(1 \ldots n)$
5. **for** $j = 1$ to l^T **do** $S[i][j] \leftarrow min_{v \in N_j} \pi_i(v)$ **end for**
6. **end for**
7. **else** {use cosine similarity *//generate random-projection signatures*}
8. **for** $i = 1$ to rb **do**
9. $rnd_i \leftarrow$ pick a random hyperplane $\in \mathbb{R}^{k^T \times 1}$
10. **for** $j = 1$ to l^T **do** $S[i][j] \leftarrow sign(P_{:j}(1) \cdot rnd_i)$ **end for**
11. **end for**
12. **end if**
13. {**Step 2. Generate hash tables**}
14. **for** $h = 1$ to b **do**
15. **for** $j = 1$ to l^T **do** $hash(S[(h-1)r + 1 : hr][j])$ **end for**
16. **end for**
17. {**Step 3. Merge clusters from hash tables** }
18. Build candidate groups: union of elements that hash to *at least one* same bucket in all hash tables, i.e. $c_1, c_2 \in g$ if $hash_h(c_1) = hash_h(c_2)$ for $\exists\ h$.
19. **for** each each candidate group g **do**
20. **while** more merges happen **do**
21. $c_r \leftarrow$ pick a random element (col. cluster) from g
22. **for all** clusters $c \in g$, $C^T(c) \neq C^T(c_r)$ **do**
23. $L^U \leftarrow$ update cost when $C^T(c)$ and $C^T(c_r)$ are merged by Equ. (2)
24. **if** $L^U < 0$ **then**
25. $l^T = l^T - 1$. $C^T(c) = C^T(c_r) = min(C^T(c), C^T(c_r))$.

26. Merge B_{ic} and B_{ic_r} $\forall i$, $1 \leq i \leq k^T$.
27. **end if**
28. **end for**
29. **end while**
30. **end for**
31. $l^{T+1} = l^T$, $C^{T+1} = C^T$

Algorithm Details. MERGE-COLCLUS, and similarly MERGE-ROWCLUS, consists of three main steps: (1) generate LSH signatures (Line 1), (2) generate hash tables (Line 13), and (3) merge clusters using hash buckets (Line 17).

In the first iteration of MERGE-COLCLUS, i.e. $T = 0$, the clusters consist of singleton nodes. As the similarity measure, we use Jaccard similarity which is high for those nodes with many exclusive common neighbors. Min-hashing is designed to capture the Jaccard similarity between binary vectors (Lines 2-6). For $T > 0$, the clusters consist of multiple nodes. As the similarity measure of two (column) clusters c_1 and c_2, we use the density similarity of their corresponding row blocks B_{ic_1} and B_{ic_2}, $\forall i$. As such, each column cluster c can be represented by a length k^T vector in which the entries denote the density $P_{ic}(1)$ of each row block i. We use their cosine similarity to compare two real-value vectors. To capture cosine similarity, we generate random-projection-based signatures (Lines 7-12). At the end of step (1), each cluster has a length-rb signature.

In step (2), we split the signature of each cluster into b length-r sub-signatures, and hash each sub-signature using standard hashing (Lines 14-16).

Step (3) involves the main merging operations. First we construct the group of candidate clusters to be merged. We put all clusters that hash to the *same* hash bucket in *at least one* hash table into the same group (Line 18). Next, we iterate over the groups to identify those clusters the merge of which will reduce the total cost (Line 19). We pick a cluster at random from a given group and test it against other clusters in the group, where we merge two clusters if the cost reduces. We continue the merges until no more merges can be done for lower cost (Lines 20-30). By focusing only on the highly similar candidate clusters within groups, MERGE-COLCLUS omits the consideration of merge between all clusters; this contributes to a reduction in the running time while enabling the merge among good candidate clusters that are highly similar.

A crucial computation in step (3) is to update the total cost when two candidate clusters are merged (Line 23). In the following we show that the update-cost L^U can be computed *locally* without requiring the re-computation of the total cost. As such, we decide to merge two clusters if their update-cost is less than 0 (Line 24), i.e. when the merge *reduces* the total objective cost.

Updating the Total Objective Cost. When two column (or row) clusters are merged, we can analyze how the encoding cost is expected to change. Without loss of generality, assume two column clusters of *sizes* c_1 and c_2 are to be merged.

Lemma 1. *If two clusters are merged, then the total cluster assignment cost, i.e.* $\sum_{j=1}^{l} c_j \log_2(\frac{m}{c_j})$*, will decrease.*

Proof. The assignment cost $c_j \log_2 \frac{m}{c_j}$ remains the same for $c_j \neq c_1, c_2$. We have

$$(c_1+c_2)\log_2\frac{m}{(c_1+c_2)} = c_1\log_2 m + c_2\log_2 m - c_1\log_2(c_1+c_2) - c_2\log_2(c_1+c_2)$$
$$< c_1\log_2 m + c_2\log_2 m - c_1\log_2 c_1 - c_2\log_2 c_2 = c_1\log_2\frac{m}{c_1} + c_2\log_2\frac{m}{c_2}. \quad \square$$

Lemma 2. *If two clusters are merged, then total cost* $\sum_{i=1}^{k}\sum_{j=1}^{l}\log_2(r_i c_j + 1)$ *of encoding the number of* 1*s for blocks will decrease.*

Proof. The $\log_2(r_i c_j + 1)$ cost remains the same for clusters $c_j \neq c_1, c_2$, $\forall i$.

$$\log_2(r_i(c_1+c_2)+1) = \log_2(r_i c_1 + r_i c_2 + 1) < \log_2(r_i^2 c_1 c_2 + r_i c_1 + r_i c_2 + 1)$$
$$= \log_2((r_i c_1 + 1)(r_i c_2 + 1)) = \log_2(r_i c_1 + 1) + \log_2(r_i c_2 + 1).$$

Lemma 3. *When two clusters merge, block encoding cost* $\sum_{i=1}^{k}\sum_{j=1}^{l} E(B_{ij})$ *will increase, i.e., if* $B_i = [B_{i1} B_{i2}]$*, then* $E(B_{i1}) + E(B_{i2}) \leq E(B_i), \forall i$.

Proof. $E(B_{ij})$ remains the same for clusters $c_j \neq c_1, c_2$, $\forall i$. We have $\forall i$,

$$E(B_i) = n(B_i) H\left(\frac{n_1(B_i)}{n(B_i)}\right) = n(B_i) H\left(\frac{n(B_{i1})P_{B_{i1}}(1) + n(B_{i2})P_{B_{i2}}(1)}{n(B_i)}\right)$$
$$\geq n(B_{i1}) H(P_{B_{i1}}(1)) + n(B_{i2}) H(P_{B_{i2}}(1)) = E(B_{i1}) + E(B_{i2})$$

where the inequality follows from the concavity of the entropy function $H(\cdot)$. (also note that $n(B_{i1}) + n(B_{i2}) = n(B_i)$). $\square$

Overall, the difference between the increase in the block encoding cost (Lemma 3) and the decrease in the cluster assignment and number of non-zeros encoding costs (Lemma 1 & 2) will determine whether two candidate clusters are merged or not (in Line 23 of Procedure 1). This difference can be computed quickly without requiring the re-computation of the total cost. Specifically, the total update-cost L^U when two (column) clusters of sizes c_1 and c_2 are merged (where there are totally m columns, and l column clusters) is equal to

$$L^U = -c_1\log_2\frac{m}{c_1} - c_2\log_2\frac{m}{c_2} + (c_1+c_2)\log_2\frac{m}{(c_1+c_2)}$$
$$\sum_i -\log_2(r_i c_1 + 1) - \log_2(r_i c_2 + 1) + \log_2(r_i(c_1+c_2)+1)$$
$$\sum_i \Big(- E(B_{i1}) - E(B_{i2}) + E(B_i) \Big) - \log^\star(l) + \log^\star(l-1) \qquad (2)$$

where the last two terms account for the difference in encoding cost of the number of column clusters, which will reduce by 1 in case of a merge.

Finally, we provide the time complexity for all steps of our method in Table 1. Details are omitted for lack of space.

Table 1. Computational complexity of CoClusLSH steps

	$T = 0$	$T \geq 1$
Step 1.	$O(n_1(\mathbf{A})rb)$	$O(klrb)$
Step 2.	$O((n+m)rb)$	$O((k+l)rb)$
Step 3.	$O((n+m)M)$	$O((k+l)M)$

3 Experiments

We next evaluate our method based on (1) clustering quality, and (2) scalability. on both real and synthetic datasets (with/without ground truth cluster labels).

3.1 Synthetic Datasets

Data Generation. To create synthetic data matrices, we use two different schemes. In the first scheme, we fix a cluster size s, and increase the number of such clusters k to obtain larger and larger graphs. In the second scheme, we fix the number of clusters, and increase the size of the clusters to obtain various size graphs. The generated matrices are diagonally strong, i.e. the density of the diagonal blocks is p, whereas the off-diagonal blocks are all-zeros. Next, we add random noise by adding ϵ fraction of non-zeros in the original matrix at random entries, to study the effect of varying noise levels on the clustering performance.

We call the first set of synthetic graphs as `CAVE1` graphs, with $s = 500$, $2 \leq k \leq 11$, $p = 0.9$, and $\epsilon = \{0.1, 0.2, 0.3, 0.4\}$. This gives us 4 sets of 10 graphs of various sizes, where each set of graphs have a different level of noise.

The second set of graphs are called `CAVE2` graphs, with $s = 50t, 1 \leq t \leq 10$, $k = 10$, and p and ϵ as before. This way we also obtain 40 graphs. We provide statistics of the largest `CAVE1` and `CAVE2` graph with 40% noise in Table 2.

Clustering Quality. We first study the effect of noise on the clustering quality. As we work with synthetic datasets, we have the ground truth for the cluster assignments and thus can compute the true/optimal encoding costs.

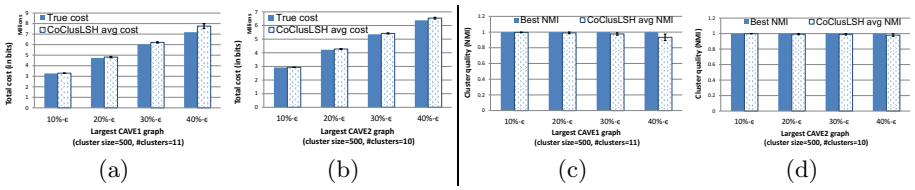

Fig. 2. (a,b) True vs. CoClusLSH cost (y-axes), (c,d) Optimal vs. CoClusLSH NMI (y-axes). (both avg. over 10 runs, bars depict σ) on the largest (a,c) `CAVE1` and (b,d) `CAVE2` graph with varying noise levels 10%-40% (x-axes).

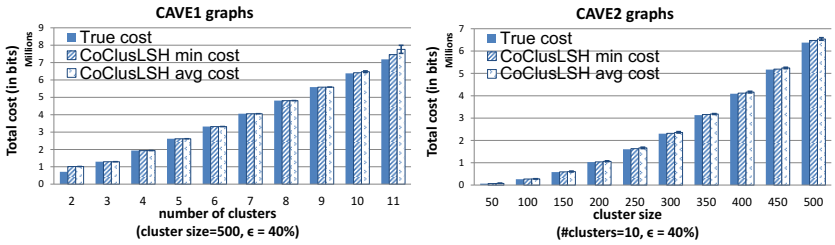

Fig. 3. True cost vs. best and avg. CoClusLSH cost (over 10 runs, bars depict σ) on all (left) `CAVE1` and (right) `CAVE2` graphs, when $\epsilon = 40\%$

In Figure 2 (a,b) we show the true cost in comparison to our CoClusLSH's cost for the largest `CAVE1` and `CAVE2` graphs, for the increasing noise levels. We observe that the gap between the optimal and CoClusLSH's cost increases with more noise, however CoClusLSH still finds good approximate solutions.

To assess the cluster assignment quality, we use the Normalized Mutual Information (NMI), a widely used measure for evaluating the clustering accuracy of a method against the ground truth clustering [18]. The ideal NMI score is 1.

Table 2. Datasets used in this work

Dataset	**A** Dim. $n \times m$	$n_1(\mathbf{A})$
US-SENATE	108 senators × 696 bills	40,609
US-HOUSE	451 rep.s × 1,646 bills	501,602
POLBLOGS	362 blogs × 5,895 words	776,870
DBLP	1,230 papers × 1,230 papers	19,267
CLASSIC	3,893 doc.s × 4,303 words	176,347
NIPS	10,617 words × 2,864 authors	160,059
YOUTUBE	77,381 users × 30,087 groups	260,240
CAVE1–40%ϵ	5,500 × 5,500	3,289,021
CAVE2–40%ϵ	5,000 × 5,000	2,974,778

Figure 2 (c,d) depicts CoClusLSH's average NMI score for the largest CAVE1 and CAVE2 graphs, for increasing noise. As before, we observe that NMI drops slightly with more noise, while it remains > 0.9 at all noise levels.

Next, we study the effect of increasing number of clusters (as in CAVE1 graphs) and of increasing cluster sizes (as in CAVE2 graphs) on the performance. Figure 3 shows the true encoding cost against CoClusLSH's best and average cost across all CAVE1 and CAVE2 graphs, respectively, for the most challenging case of $\epsilon = 40\%$. We observe that CoClusLSH recovers a low cost solution in all cases.

Scalability. Next, we study the growth in the running time of CoClusLSH with increasing graph size. In §2.3 we showed that the running time is proportional to the number of nonzeros, the number of rows and columns, and the maximum number of clusters that hash to the same group. In Figure 4, we show the total time w.r.t. these parameters, for all CAVE1 and CAVE2 graphs, at varying noise levels. We observe that the run time grows linearly, and that more noise demands more time for our algorithm to converge.

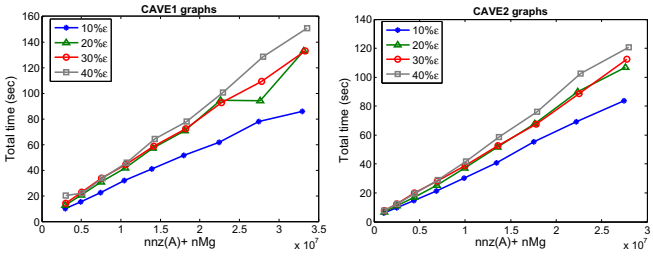

Fig. 4. Total running time (in sec) of our CoClusLSH on (left) CAVE1 and (right) CAVE2 graphs with varying noise

3.2 Real Datasets

Data description. Our real-world datasets include (Table 2): US-SENATE with senators and US-HOUSE with The House representatives voting (1 'yes', 0 'no') on congressional bills in the 111th US Congress; POLBLOGS with political blogs and the words they use; DBLP with academic papers and their commonly used terms relations; and finally CLASSIC with documents and the words they contain.

For the five datasets described above, we have the ground truth labels but only for the rows. In particular US-SENATE and US-HOUSE both consist of two

classes; (1) liberal and (2) conservative congressmen, POLBLOGS also has two classes; (1) liberal and (2) conservative blogs, DBLP contains papers from four classes of venues; (1) SIGIR (information retrieval), (2) STOC+FOCS (theory), (3) AAAI (artificial intelligence), and (4) TODS (database systems), and finally CLASSIC consists of documents from three different classes; (1) MEDLINE (medical journals), (2) CISI (information retrieval) and (3) CRANFIELD (aero-dynamics). In addition, we used two more real datasets, namely NIPS and YOUTUBE, with many rows and columns for our scalability experiments. Unfortunately, there are no class labels for the rows or columns for these datasets.

Clustering Quality. We evaluate CoClusLSH's clustering performance using two clustering quality measures, purity and NMI as before [18], using the ground truth labels of the row nodes (note that we do not have the ground truth labels for the column nodes for our datasets, thus we cannot compute the optimal encoding cost as for the synthetic datasets).

Purity measures the coherence of labels within each cluster. It, however, is expected to increase with the number of output clusters—in the extreme case where each node belongs to its own cluster, purity becomes 1. NMI can trade off the quality of the clustering against the number of clusters.

We report both measures and the number of clusters CoClusLSH outputs on our real datasets in Table 3. We observe that CoClusLSH does particularly well on the US-SENATE and US-HOUSE datasets, while clustering accuracy in comparison is lower for the other datasets. Looking at the output clusters by CoClusLSH, we realize that the cluster structure in the congress datasets is well pronounced, while the rest of the real datasets are quite sparse with no clear cluster structure. We show an example output of CoClusLSH on the US-HOUSE and DBLP datasets in Figure 5, where CoClusLSH performs well on recovering salient cluster structure.

Table 3. CoClusLSH clustering quality (purity and NMI) on real datasets (avg. over 10 runs, all standard deviations were < 0.03). Also shown is k^*, avg. number of clusters found and k, the true number of clusters.

Dataset	*Purity* (avg)	*NMI* (avg)	k^* (avg)	k (true)
US-SENATE	0.9960	0.5569	12.0	2
US-HOUSE	0.9934	0.5172	28.9	2
POLBLOGS	0.5539	0.0142	18.9	2
DBLP	0.4723	0.0949	6.6	4
CLASSIC	0.3987	0.0241	5.0	3

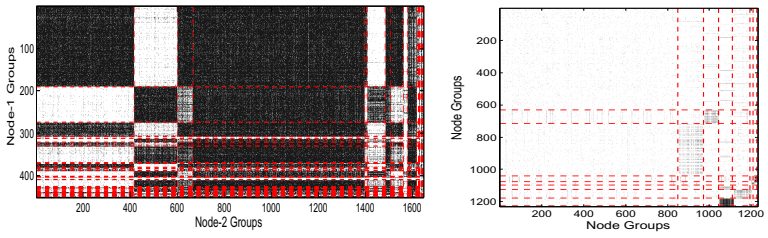

Fig. 5. Adjacency matrix of (left) US-HOUSE and (right) DBLP, with rows and columns arranged by the cluster assignments of CoClusLSH.

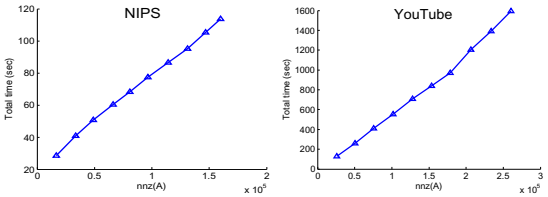

Fig. 6. Total running time of CoClusLSH on growing (left) `NIPS` and (right) `YOUTUBE` graphs.

Scalability. Finally, we also experimentally study that the running time of CoClusLSH with respect to the input size on real graphs. For running time measurements, we use the `NIPS` and `YOUTUBE` datasets with many rows each. To generate graphs of growing size, we increasingly sample the rows of these matrices, and report average running time over 10 runs in Figure 6. We observe that the running time grows linearly with the input graph size.

4 Related Work

Clustering algorithms in the row-mode *only* include k-means and its parameter-free variants [13,23], spectral [21], and (probabilistic) hierarchical clustering [15,20,27]. Our problem deals with *simultaneous* clustering of rows and columns, known as bi(dimensional)-, co-, or block clustering. Information-theoretic co-clustering [9] employs a lossy coding scheme to co-cluster a two-dimensional joint probability distribution, however, it requires the number of clusters as input. Conjunctive clustering [19] finds top-k largest bi-clusters in bipartite graphs, but requires two parameters for lower-bounding cluster sizes in each dimension, a diversity parameter controlling overlap, and a density parameter.

Hierarchical tiling [12] extracts nested tiles (or blocks) of various densities in the adjacency matrix. While parameter-free,it involves a quadratic preprocessing step for row/column re-ordering. Bi-clustering has also been explored in bioinformatics [1,24] often requiring the number of clusters as user input.

Cross-associations [7] automatically selects the number of clusters and scales well to large graphs. It has been used in graph partitioning [6], and extended to time-evolving [28] and attributed graphs [4]. Our approach exhibits the same merits as [7], while enabling the cluster hierarchies.

Other relevant work to ours include frequent itemset and association rules mining [3,5,14], where the support parameter is critical. In information retrieval LSI [8] uses SVD to find latent concepts in the data matrix, which requires the number of hidden concepts. In addition, subspace clustering [2,17] aims at finding all the dense clusters in all subspaces. These methods often take input parameters such as density and size thresholds to quickly scan their search space (also see [16] for a survey on subspace, correlation, and pattern-based clustering).

Finally, LSH has been used to accelerate several other problems, such as similarity search [26], outlier detection [29], and k-nearest neighbor search [22].

5 Conclusion

We propose an approach for co-clustering large binary matrices, based on an information-theoretic objective. To solve our objective, we develop a fast, bottom-up clustering algorithm that rapidly finds the most similar rows/columns to merge using locality-sensitive hashing, and determines the number of clusters automatically. We demonstrate the effectiveness and scalability of the proposed approach on real and synthetic datasets, where our algorithm recovers the cluster structure with high accuracy, and has a running time that grows linearly with the input matrix size. Our source code is freely available for academic use.[2]

Acknowledgements. This material is based upon work supported by the Army Research Office under Contract No. W911NF-14-1-0029 and Stony Brook University Office of Vice President for Research. Any findings and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties.

References

1. Abdullah, A., Hussain, A.: A new biclustering technique based on crossing minimization. Neurocomputing 69(16-18), 1882–1896 (2006)
2. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data. In: SIGMOD, pp. 94–105 (1998)
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD 22(2), 207–216 (1993)
4. Akoglu, L., Tong, H., Meeder, B., Faloutsos, C.: Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In: SDM (2012)
5. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) PKDD 2002. LNCS (LNAI), vol. 2431, pp. 74–86. Springer, Heidelberg (2002)
6. Chakrabarti, D.: AutoPart: Parameter-free graph partitioning and outlier detection. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 112–124. Springer, Heidelberg (2004)
7. Chakrabarti, D., Papadimitriou, S., Modha, D.S., Faloutsos, C.: Fully automatic cross-associations. In: ACM SIGKDD, pp. 79–88 (2004)
8. Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: Indexing by latent semantic analysis. JASI 41(6), 391–407 (1990)
9. Dhillon, I., Mallela, S., Modha, D.: Information- theoretic co-clustering. In: ACM SIGKDD (2003)
10. Fortunato, S., Barthélemy, M.: PNAS, 104(1), 36 (2007)
11. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
12. Gionis, A., Mannila, H., Seppänen, J.K.: Geometric and combinatorial tiles in 0-1 data. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 173–184. Springer, Heidelberg (2004)

[2] CoClusLSH code: `http://www.cs.stonybrook.edu/~leman/pubs.html#code`

13. Hamerly, G., Elkan, C.: Learning the k in k-means. In: NIPS (2003)
14. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining top-k frequent closed patterns without minimum support. In: ICDM, pp. 211–218 (2002)
15. Karypis, G., Han, E.-H., Kumar, V.: Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer 32(8) (1999)
16. Kriegel, H.-P., Kröger, P., Zimek, A.: Clustering high-dimensional data: A survey. TKDD 3(1), 1:1–1:58 (2009)
17. Kröger, P., Kriegel, H.-P., Kailing, K.: Density-connected subspace clustering for high-dimensional data. In: SDM (2004)
18. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
19. Mishra, N., Ron, D., Swaminathan, R.: On finding large conjunctive clusters. In: Schölkopf, B., Warmuth, M.K. (eds.) COLT/Kernel 2003. LNCS (LNAI), vol. 2777, pp. 448–462. Springer, Heidelberg (2003)
20. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Physical Review E 69 (2004)
21. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: NIPS (2001)
22. Pan, J., Manocha, D.: Bi-level locality sensitive hashing for k-nearest neighbor computation. In: ICDE, pp. 378–389 (2012)
23. Pelleg, D., Moore, A.: X-means: Extending K-means with efficient estimation of the number of clusters. In: ICML (2000)
24. Reiss, D.J., Baliga, N.S., Bonneau, R.: Integrated biclustering of heterogeneous genome-wide datasets. BMC Bioinformatics 7, 280 (2006)
25. Rissanen, J.: A universal prior for integers and estimation by minimum description length. The Annals of Statistics 11(2), 416–431 (1983)
26. Satuluri, V., Parthasarathy, S.: Bayesian locality sensitive hashing for fast similarity search. PVLDB 5(5), 430–441 (2012)
27. Slonim, N., Tishby, N.: Agglomerative information bottleneck. In: NIPS (1999)
28. Sun, J., Faloutsos, C., Papadimitriou, S., Yu, P.S.: Graphscope: parameter-free mining of large time-evolving graphs. In: ACM SIGKDD, pp. 687–696 (2007)
29. Wang, Y., Parthasarathy, S., Tatikonda, S.: Locality sensitive outlier detection: A ranking driven approach. In: ICDE, pp. 410–421 (2011)