Discovering Hidden Relationships via Efficient Co-clustering of Sparse Matrices

(Work in progress – Tuesday 15th October, 2013 @ 18:50)

Brian Thompson† Linda Ness‡ David Shallcross‡ Devasis Bassu‡

Abstract

We present a new approach for discovering hidden relationships in sparse bipartite data by identifying large, dense biclusters in the corresponding matrix. We motivate a new class of metrics to measure the quality of a bicluster partition, and compare them to existing metrics. We then present a heuristic algorithm that efficiently searches the space of possible co-clusterings for one which maximizes the value of a given metric. We evaluate our approach with experiments on synthetic and real-world datasets.

1 Introduction

1.1 Background and Motivation Bipartite data consists of two sets of objects and the relationships between them. This can be visualized as a (optionally weighted) bipartite graph, as in Figure 1(a), and in practice is commonly stored as a matrix $M$ with rows and columns corresponding to the two sets, respectively, as in Figure 1(b). In many real-world domains, it is of interest to find biclusters – subsets of row and column objects – with many strong pairwise relationships, which would appear as dense rectangular blocks in the matrix. However, since the order of the rows and columns of a matrix are arbitrary, such structure may not be readily apparent. Simultaneously permuting and clustering the rows and columns of the matrix – a task known as co-clustering – may reveal relationship structure that was hidden under the original orderings.

For example, if the rows of $M$ are people and the columns are restaurants, an example of a dense bicluster could be the set of people who are lovers of Italian cuisine. If $M$ is a document-term matrix, biclusters could indicate bodies of literature on similar topics. In collaborative filtering, the discovered biclusters can drive prediction algorithms for consumer or user preferences.

The general nature of the problem and the size and sparsity of typical real-world relationship data motivate our search for an efficient co-clustering algorithm to automate the simultaneous discovery of multiple dense biclusters in sparse matrices. We seek an algorithm that does not rely on domain-specific knowledge or data-specific parameters so it can be easily applied across domains. Before presenting our own approach, we provide a survey of related work.

1.2 Related Work There is a significant amount of related work in the data mining, machine learning, databases, and bioinformatics literature – each motivated by different applications and with different goals in mind. These tasks may be variously referred to in the literature as co-clustering, biclustering, two-mode clustering, or matrix block partitioning.

In bioinformatics, biclustering algorithms have been developed to find patterns in gene expression [9, 13, 3]. A matrix is constructed where each row is a gene, each column is an experiment under different conditions, and each entry indicates the level of expression of that gene under those conditions. Matrices are dense and real-valued, and the goal is to identify genes that behave similarly, demonstrated by expression levels that have a constant, linear, or multiplicative relationship across similar experimental conditions. Due to the density of the matrices and the more intricate bicluster structure desired, algorithms in this domain tend to find only one bicluster at a time, and efficiency concerns frequently restrict their use to smaller datasets.

Algorithms for finding structure in matrices can be used
Navathe et al. address the problem of vertical partitioning of a database, which groups attribute columns that frequently need to be accessed together [11]. Muthukrishnan et al. consider the problem of rectangular partitioning of a matrix, partitioning the matrix into a set of rectangular tiles without permuting the rows or columns, which is useful for maintaining multi-dimensional histograms [10]. They consider a variety of evaluative metrics, and give heuristics and complexity theoretic results for several related optimization problems. For the applications we are interested in, row and column permutations are allowed, and are frequently necessary to discover latent structure.

Co-clustering has also been found to improve upon standard techniques for clustering uni-partite data. Such approaches entail first computing a similarity matrix, a symmetric square matrix indicating the similarity of each pair of points in the dataset, to which co-clustering algorithms are then applied. A similar approach can be employed for graph partitioning, where the goal is to cluster the vertices in the graph so that there is a high density of edges within clusters and few edges going between them, a task which has applications to community discovery and parallel computing for graph algorithms. Here, co-clustering algorithms would be applied to the adjacency matrix.

In these scenarios, co-clustering is performed on a square matrix, where the rows and the columns represent the same set of objects. The desired result is a partitioning of the objects, which corresponds to a block diagonal structure in the matrix, like that seen in Figure 2(a). Numerous algorithms to achieve this have been proposed in the literature based on linear algebraic dimensionality reduction techniques, referred to broadly as spectral clustering [8]. Further works have suggested algorithms which permit a slightly more general matrix structure, such as block diagonal with overlap [5], block tridiagonal [12], or rectangular matrices [6]. In some domains, however, matrices arise with more varied block structures, such as that in Figure 2(b), which would not be captured well by these methods. Another approach is to first define a metric over co-clusterings, and then search for a co-clustering which optimizes the metric. Dhillon et al. define a metric based on mutual information [2]. Their algorithm optimizes the metric subject to a constraint on the number of row and column clusters. On the other hand, our approach is parameter-free, searching over partitions with varying numbers of clusters and implicitly determining the number of clusters that yields the best result. Chakrabarti et al. follow a similar approach, but are motivated by the task of matrix compression, and propose encoding cost as a metric, the number of bits required to represent a matrix [1]. They present the Cross-Association algorithm, a heuristic for finding a co-clustering with minimal encoding cost, incrementally increasing the number of row or column clusters until a local optimum is reached. It is parameter-free, a strength over approaches which require the number of clusters to be known in advance. However, the efficiency of their algorithm depends on quick convergence to a small number of clusters. This may be effective for achieving their objective of a good compression ratio, but may run contrary to our goal of finding dense biclusters, especially in the case of very large and sparse matrices. Furthermore, we found that in practice, since the algorithm alternates between refining the row and column clusters instead of doing both simultaneously, it may get stuck at a local optimum because neither the rows nor the columns alone can sufficiently distinguish clusters. A simple example of this is a matrix where all rows have the same density, and likewise for the columns – the algorithm will never progress past a single row and column cluster.

1.3 Contributions and Outline

Our contributions can be summarized as follows:

- We motivate two desirable properties of co-clustering metrics that aim to reward large, dense biclusters.
- We propose a class of co-clustering metrics which uniquely satisfy those properties among known metrics.
- We introduce the CC-MACS (Co-Clustering via Maximal Anti-Chain Search) algorithm, an efficient heuristic algorithm to find a good co-clustering of an $m \times n$ matrix in $O \left( N \cdot \max \left( \log(mn), \log^2 \left( \frac{mn}{N} \right) \right) \right)$ time, where $N$ is the number of non-zeros in the matrix.

In this work, we break away from the traditional mindset that a good co-clustering means a small number of clusters. Instead of having a small number of very large blocks, CC-MACS may return a co-clustering with hundreds or thousands of blocks, among which will be the dense biclusters we are most interested in. To our knowledge, this idea is
novel to our approach, and allows us to out-perform techniques based on low-dimensional approximations or requiring the number of clusters to be specified in advance. The CC-MACS algorithm is also designed to leverage the sparsity of many real-world datasets, running in time sub-linear in the size of the matrix for sparse matrices.

In particular, our methods have the following benefits over previously proposed approaches: (1) the dense biclusters in the matrix need not have a block diagonal structure; (2) our algorithm explores the breadth of the search space rather than getting stuck at local optima; (3) the results are not dependent on user-specified parameters; and (4) our algorithm is sub-linear in the size of the matrix for \( N \ll mn \), making it extremely efficient for large, sparse datasets.

In Section 2.1, we give preliminary definitions and the framework for our approach. Section 2.2 addresses the question of choosing an appropriate metric. We present the CC-MACS (Co-Clustering via Maximal Anti-Chain Search) algorithm in Section 2.3 and analyze its run time in Section 2.4. In Section 3, we evaluate our approach with experiments on synthetic and real-world datasets. In Section 4, we discuss the significance of our work and offer directions for future work.

2 Methods

2.1 Definitions and Framework

Let \( M \) be an \( m \times n \) matrix. A bicluster of \( M \) is a subset of matrix entries formed by the intersection of a set of rows \( I \subseteq [m] \) and a set of columns \( J \subseteq [n] \), and is denoted by \( M_{I,J} \). We define the weight of a bicluster \( B = M_{I,J} \) to be \( w(B) = \sum_{i \in I, j \in J} M_{i,j} \); the area \( a(B) = |I| \cdot |J| \); the semiperimeter \( s(B) = |I| + |J| \); and the density \( d(B) = w(B)/a(B) \).

Figure 3 shows two different biclusters in the same matrix. Bicluster \( B_2 \) is arguably better than \( B_1 \) since it is the same size but more dense, indicating a stronger association between the corresponding rows and columns.

Let \( \beta(M) \) denote the set of all possible biclusters of \( M \). A bicluster partition of \( M \) is a set of biclusters \( \Pi \subseteq \beta(M) \) such that each element \( M_{i,j} \) is contained in exactly one bicluster. Co-clustering is the data mining task of simultaneously clustering the rows and columns of \( M \), which naturally corresponds to a bicluster partition of \( M \).

Our approach consists of two main components: (1) define a quality metric for bicluster partitions; and (2) find a co-clustering that maximizes the value of the metric. We address each of these tasks in the following sections.

2.2 Choosing a Metric

Here we consider metrics to evaluate the quality of a bicluster partition. A bicluster partition metric \( \mu \) is a mapping from bicluster partitions to real values, i.e. \( \mu(\Pi) \in \mathbb{R} \) where \( \Pi \) is a bicluster partition.

A variety of bicluster partition metrics have been proposed in the literature. To decide which are most appropriate for our context, we first suggest two desirable properties, motivated by our goal of identifying large, dense biclusters. Figure 4 gives motivating examples of properties \( P_1 \) and \( P_2 \).

\( P_1 \) Merging a positive-weight bicluster with a zero-weight bicluster decreases the value of the metric.

\( P_2 \) Merging two non-empty biclusters of the same density increases the value of the metric.

We now propose a class of metrics that satisfy both properties:

\[
\mu_\alpha(\Pi) = \sum_{B \in \Pi} a(B)^2 \cdot d(B)^{2+\alpha} : \alpha \geq 0
\]

The intuition is that the first term favors larger biclusters and the second term favors denser biclusters, so overall the metric favors partitions containing biclusters that are both large and dense. The value of the parameter \( \alpha \) can be used to balance the trade-off between size and density of the biclusters.

**THEOREM 2.1.** For all \( \alpha \geq 0 \), \( \mu_\alpha \) satisfies property \( P_1 \).

**THEOREM 2.2.** For all \( \alpha \geq 0 \), \( \mu_\alpha \) satisfies property \( P_2 \).

A proof of these two theorems can be found in the supplementary materials.

We now examine which of these properties are satisfied by several previously proposed and baseline metrics in Table 1. That none of the previously proposed metrics examined here satisfy both properties does not imply that they are bad metrics. Rather, they were proposed with different objectives in mind. Properties \( P_1 \) and \( P_2 \) were motivated by our goal of finding large, dense biclusters. Other objectives could be optimal compression of the matrix (inverse encoding cost) or non-uniformity of densities of the biclusters (Kullback-Leibler divergence from uniform).

<table>
<thead>
<tr>
<th>Metric</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverse encoding cost [1]</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Kullback-Leibler divergence from uniform</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( \sum_{B \in \Pi} w(B) )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( \sum_{B \in \Pi} d(B) )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( \sum_{B \in \Pi} w(B) \cdot d(B) )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( \sum_{B \in \Pi} (a(B)^2/s(B)) \cdot d(B)^{2+\alpha} )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>

Table 1: Properties satisfied by several metrics.

We note that our proposed metrics take the form \( \sum_{B \in \Pi} f(B) \), where \( f : \beta(M) \rightarrow \mathbb{R} \) is a function of the weight and dimensions of a bicluster. The \( \sum \) operator is both commutative and associative, so our metrics can be computed by combining the \( f \)-values for the biclusters in any order. In the next section, we leverage this property to develop more efficient algorithms to find good bicluster partitions.
2.3 The CC-MACS Algorithm

We now present the CC-MACS (Co-Clustering via Maximal Anti-Chain Search) algorithm, which efficiently searches for a good co-clustering according to a given metric. We first note that the total number of possible co-clusterings of an $m \times n$ matrix is exponential in the size of the matrix (the product of the $m$th and $n$th Bell numbers), so an exhaustive search is infeasible. Our strategy for overcoming this computational challenge is to first build trees on the rows and columns, respectively, and then to consider only co-clusterings corresponding to maximal anti-chains in the trees.

A maximal anti-chain of a rooted tree is a maximal set of nodes in the tree, none of which is a descendant of any other. For example, the blue nodes in each of the trees in Figures 5(c)-(i) form a maximal anti-chain. Note that the subtrees of the nodes in a maximal anti-chain correspond to a partition of the leaves of the tree. Therefore any pair of maximal anti-chains of the row and column trees, respectively, corresponds to a co-clustering of the matrix.

This is still a computational challenge, however, because there are $\Omega(2^n)$ maximal anti-chains in a complete binary tree with $n$ leaves. We employ a heuristic to find the most likely candidates by traversing the row and column trees simultaneously, starting at the leaves and greedily merging the nodes that result in the greatest increase in the metric value. Figure 5 illustrates an example run of the CC-MACS algorithm. Pseudocode is given in Algorithm 2.1.

**Algorithm 2.1.** The CC-MACS Algorithm

**Input:** An $m \times n$ matrix $M$ and a bicluster partition metric $\mu(\Pi) = \bigoplus_{B \in \Pi} f(B)$, where $\bigoplus$ is a commutative and associative binary operator, and $f : \beta(M) \rightarrow \mathbb{R}$ is a function of the weight and dimensions of a bicluster.

**Output:** A co-clustering $\Pi$ of $M$.

1. Initialize partitions $\Pi^{row}$ and $\Pi^{col}$ to be the sets of singletons of the rows and columns of $M$, respectively.
2. Construct a $k$-d tree $T^{row}$ over the vector sums corresponding to row clusters in $\Pi^{row}$, after first applying a random projection, as in [4]. Let $L(T^{row})$ denote the set of leaves of $T^{row}$, and let $I_x \subseteq [m]$ denote the indices of rows in the subtree rooted at node $x \in T^{row}$.
3. Construct a $k$-d tree $T^{col}$ over the vector sums corresponding to column clusters in $\Pi^{col}$, similarly to above. Let $L(T^{col})$ denote the set of leaves of $T^{col}$, and let $J_y \subseteq [n]$ denote the indices of columns in the subtree rooted at node $y \in T^{col}$.
4. Populate a two-dimensional array $W$ indexed by nodes in $T^{row}$ and $T^{col}$, respectively, where entry $W[x,y]$ is the number of non-zeros in the bicluster $M_{I_x,J_y}$. From this, populate another two-dimensional array $F$ of the same dimensions, containing values $F[x,y] = f(M_{I_x,J_y})$. 

**Figure 3:** Two biclusters in a matrix.

**Figure 4:** Examples illustrating properties (a) $P_1$ and (b) $P_2$. 

2.3 The CC-MACS Algorithm We now present the CC-MACS (Co-Clustering via Maximal Anti-Chain Search) algorithm, which efficiently searches for a good co-clustering according to a given metric. We first note that the total number of possible co-clusterings of an $m \times n$ matrix is exponential in the size of the matrix (the product of the $m$th and $n$th Bell numbers), so an exhaustive search is infeasible. Our strategy for overcoming this computational challenge is to first build trees on the rows and columns, respectively, and then to consider only co-clusterings corresponding to maximal anti-chains in the trees.

A maximal anti-chain of a rooted tree is a maximal set of nodes in the tree, none of which is a descendant of any other. For example, the blue nodes in each of the trees in Figures 5(c)-(i) form a maximal anti-chain. Note that the subtrees of the nodes in a maximal anti-chain correspond to a partition of the leaves of the tree. Therefore any pair of maximal anti-chains of the row and column trees, respectively, corresponds to a co-clustering of the matrix.

This is still a computational challenge, however, because there are $\Omega(2^n)$ maximal anti-chains in a complete binary tree with $n$ leaves. We employ a heuristic to find the most likely candidates by traversing the row and column trees simultaneously, starting at the leaves and greedily merging the nodes that result in the greatest increase in the metric value. Figure 5 illustrates an example run of the CC-MACS algorithm. Pseudocode is given in Algorithm 2.1.
Figure 5: Illustration of the CC-MACS algorithm. Figure (a) shows an example matrix as input to the algorithm. Figure (b) shows the same matrix permuted to reflect the k-d trees constructed in Steps 2 and 3. The maximal anti-chains $S^{row}$ and $S^{col}$ are indicated by the blue nodes, initially set to be the leaves in Step 5 (Figure (c)), and dynamically updated during the loop in Step 8 (Figures (d)-(g)). Figure (h) shows the co-clustering after the final iteration of the loop, when $S^{row}$ and $S^{col}$ are the roots of the trees, corresponding to the single bicluster consisting of the entire matrix. Figure (i) shows the result of the CC-MACS algorithm, the co-clustering that was found to maximize the metric value, indicated by the red lines.
5. Let $S^{row}_{max}$, $S^{row}_{min}$ and $S^{col}_{max}$, $S^{col}_{min}$ be maximal anti-chains over $T^{row}$ and $T^{col}$, respectively. Initialize them as $S^{row} = S^{row}_{max} = L(T^{row})$, $S^{col} = S^{col}_{max} = L(T^{col})$.

6. Maintain the current and maximum metric values, $\mu_{cur}$ and $\mu_{max}$, and initialize them for the partition corresponding to the current $S^{row} \times S^{col}$.

7. Maintain a max heap $H^{row}$ containing only nodes $x \in T^{row}$ such that $x.left, x.right \in S^{row}$, with priorities $h^{row}(x) = \sum_{y \in S^{col}} F[x, y] - f[x.left, y] - f[x.right, y]$, the marginal value from including $x$ in a maximal anti-chain instead of its children. Construct $H^{col}$ similarly.

8. While at least one of $H^{row}$ and $H^{col}$ is non-empty:
   - Without loss of generality, suppose that $H^{row}.maxPriority(\cdot) \geq H^{col}.maxPriority(\cdot)$.
   - Update the dynamic data structures and variables:
     \[ x := H^{row}.deletemax(\cdot); \]
     \[ S^{row} := S^{row} + x - x.left - x.right; \]
     \[ \mu_{cur} := \mu_{cur} + h^{row}(x); \]
     Update $H^{col}(\cdot)$ for each $y \in H^{col}$;
   - If $x.sibling \in S^{row}$, do $H^{row}.add(x.parent)$;
   - If $\mu_{cur} \geq \mu_{max}$, perform the following updates:
     \[ \mu_{max} := \mu_{cur}; S^{row} := S^{row}; \]

9. Update $\Pi^{row}$ and $\Pi^{col}$ to be the row and column partitions corresponding to nodes in $S^{row}_{max}$ and $S^{col}_{max}$, respectively.

10. Repeat steps 2-9 using the updated $\Pi^{row}$ and $\Pi^{col}$. Continue iterating while at least one is updated.

11. Return $\Pi = \Pi^{row} \times \Pi^{col}$, the co-clustering formed by the intersection of $\Pi^{row}$ and $\Pi^{col}$.

### 2.4 Running Time
Consider an $m \times n$ matrix containing $N$ non-zero entries. Step 1, initializing the partitions, takes $O(m+n)$ time. The $m$ rows can be projected onto a $\log(m)$-dimensional space in $O(N \log m)$ time, after which the k-d tree on the rows can be computed in $O(m \log m)$ time; likewise, the $n$ rows can be projected onto a $\log(n)$-dimensional space in $O(N \log n)$ time, after which the k-d tree on the columns can be computed in $O(n \log n)$ time; so Steps 2 and 3 take $O(N \cdot (\log(m + \log n))) = O(N \log(mn))$ time total. Using dynamic programming, the arrays $W$ and $F$ in Step 4 can be populated in time linear in the number of non-zeros in the resulting arrays, which is $O(N \cdot \log^2 \frac{mn}{N})$ (see supplementary materials for proof). We note that this bound is tight (ibid.). Step 5 is $O(m+n)$, and Step 6 is $O(N)$. Computing the priority values in Step 7 takes $O(N)$ time. Inserting and deleting the elements in the row and column heaps in Steps 7-8 takes $O(m \log m + n \log n)$ time total since each node is inserted and deleted at most once. Updating $h^{col}(\cdot)$ in Step 8 takes $O(\log n)$ time for each $y \in H^{col}$, which is performed for each iteration of the loop where an element $x \in H^{row}$ was chosen such that $F[x, y]$ is non-zero, for a total of $O(N \log n)$; the total for iterations where an element from $H^{col}$ was chosen is $O(N \log m)$. In total, the CC-MACS algorithm runs in $O(\kappa \cdot N \cdot \max \{(\log(mn), \log^2 \frac{mn}{N})\})$ time, where $\kappa$ is the number of iterations of Step 10.

### 3 Evaluation
We compare the effectiveness of the CC-MACS algorithm for finding dense biclusters with existing and baseline co-clustering algorithms. These are the algorithms we use in our experiments:

- CC-MACS algorithm with $\mu_2 = \sum_{B \in \Pi} \frac{a(B)^2}{\pi(B)} \cdot d(B)^4$
- CC-MACS algorithm with $\mu_1 = \sum_{B \in \Pi} \frac{a(B)^2}{\pi(B)} \cdot d(B)^3$
- CC-MACS algorithm with $\mu_0 = \sum_{B \in \Pi} \frac{a(B)^2}{\pi(B)} \cdot d(B)^2$
- Cross-Association: minimizes encoding cost [1]
- One-block: consists of a single large bicluster
- Singletons: each matrix entry is in its own bicluster

#### 3.1 Prediction Accuracy
First we outline an unsupervised learning task by which to evaluate co-clustering algorithms. Given a matrix $M$ containing an unknown set of possibly noisy biclusters $\{B_k\}_{k \in \mathbb{N}}$, co-cluster $M$ such that elements of bicluster $B_k$ only appear in the same block in the partition as other elements of $B_k$. That is, the pair of matrix entries $(a_1, a_2)$ form a positive instance if $\exists k \in \mathbb{N}$ such that $a_1, a_2 \in B_k$; and a negative instance if at least one of $a_1, a_2$ appears in a bicluster and $\exists k \in \mathbb{N}$ such that $a_1, a_2 \not\in B_k$.

We perform experiments to evaluate the accuracy of several co-clustering algorithms for the learning task described above using synthetically generated matrices with a variety of parameters. Specifically, MGEN$(m, n, k, r, s, p)$ generates an $m \times n$ matrix $M$ with $k$ biclusters of size $r \times s$ selected randomly from $M$, where each non-bicluster entry is a 0, and each bicluster entry is a 1 with probability $1 - p$. For each co-clustering algorithm, we compute the precision, recall, and $F_1$-score (a statistical measure that considers both precision and recall), averaged over 10 trials.

Figure 6 shows the results when we vary the number of biclusters. We fix $m = n = 1024, r = s = 4,$ and $p = 0$, and let $1 \leq k \leq 256$. The Singletons method has perfect precision since there are no false positives, but 0 recall; on the other hand, the One-block method has perfect recall but close to 0 precision. The CC-MACS algorithm with the $\mu_2$ and $\mu_1$ metrics out-perform the other algorithms in finding.
a good balance between precision and recall, as measured by the $F_1$-score. Performance inevitably deteriorates as the number of biclusters increases, as they are more likely to overlap and create conflicts in finding a hard co-clustering.

In Figure 7, we fix $m = n = 1024$, $k = 16$, and $p = 0$, and vary the size of the biclusters from $1 \times 1$ to $32 \times 32$. The results are similar to the previous experiment, with performance reaching a peak around $2 \times 2$ or $4 \times 4$, and declining as the size of the biclusters increases (for smaller $k$, the peak would be later).

Figure 8 analyzes robustness to missing values. We fix $m = n = 1024$, $k = 8$, and $r = s = 16$, and let $0 \leq p \leq 0.5$. Results show that the CC-MACS algorithm with the $\mu_2$ metric still achieves good precision and recall with up to 30% missing.

### 3.2 Real-world Data

We now apply the CC-MACS algorithm to find dense biclusters in real-world matrices. First we look to the domains of finite element modeling and quantum chemistry. These datasets were obtained from the NIST Matrix Market repository, and chosen because of their clear block structure. Some of the matrices contain complex-valued entries. To accommodate this, in the following experiments, we treat all data as $\{0, 1\}$-matrices, where a 1 indicates the presence of a non-zero value.

Figure 9 shows the results from the Cross-Association and CC-MACS algorithms. In particular, the CC-MACS algorithm with the $\mu_2$ metric is seen to be effective at identifying large, dense biclusters in these datasets. The $\mu_0$ metric returned the trivial single-block co-clustering on all datasets, reflecting that it does not put enough weight on the density of a bicluster. The $\mu_1$ and $\mu_2$ metrics, on the other hand, seem to have performed quite well – in fact, they achieve a lower encoding cost than the co-clustering output by the Cross-Association algorithm itself. The Cross-Association algorithm may have perform poorly on these datasets either because it finds a local optimum before the global optimum is reached, or because in sparse matrices the encoding cost may be minimized by having a single bicluster containing the entire matrix.

Next we perform experiments on the Meme-Tracker dataset, a large collection of memes extracted from the web by Leskovec et al. [7]. To study the dynamics of the news cycle, they processed text from hundreds of thousands of blogs and news websites to extract distinguishable phrases that suddenly appeared with unusual frequency. They classified these into phrase clusters – each representing a meme – to account for variations in spelling, truncation, and other modifications that may occur as a phrase spreads through the web. From this data, we constructed a binary matrix where each row corresponds to one of the memes, each column corresponds to a web domain, and entry $(i, j)$ is a 1 if and only if the $i$th meme was mentioned on a website at the $j$th domain. After filtering out domains with less than 10 memes, the resulting matrix has 71,566 rows; 47,228 columns; and 4,026,266 non-zero entries (0.1% of the matrix). We will refer to this as the MT matrix.

We ran the CC-MACS algorithm on the MT matrix using the $\mu_0$, $\mu_1$, and $\mu_2$ metrics. The $\mu_0$ metric returned the trivial single-block co-clustering, as in the previous experiment, but the $\mu_1$ and $\mu_2$ metrics yielded more illuminating results. Table 2 shows some of the top biclusters found.

## 4 Significance and Impact

We have presented a new approach for discovering hidden relationships in bipartite data. The literature on co-clustering spans multiple disciplines, but the variety in applications and characteristics of the data motivate different goals and therefore require different approaches. Our interest is in finding dense biclusters in very large and sparse datasets. We proposed a heuristic algorithm that leverages the sparsity of the data to efficiently traverse the space of possible co-clusterings for one that maximizes a given metric.

Furthermore, while there are a number of existing metrics, we note that different metrics may be appropriate for different applications. We suggested two intuitive properties that a metric for finding large, dense biclusters should satisfy, and proposed a new class of metrics that satisfy them both. We found that our co-clustering algorithm coupled with our proposed metrics compared favorably to state-of-the-art and baseline methods for a classification task on synthetic data, and effectively identified large, dense biclusters in several real-world datasets.

Possible directions for future work include considering a more general class of bicluster partitions, not just those formed by a co-clustering of the rows and columns; providing bounds on the approximation factor of our heuristic algorithm; or adapting our approach for distributed computation.

## References


Figure 6: Precision, recall, and $F_1$-score of several algorithms as the number of biclusters varies.

Figure 7: Precision, recall, and $F_1$-score of several algorithms as the size of biclusters varies.

Figure 8: Precision, recall, and $F_1$-score of several algorithms as the percentage of missing values varies.

<table>
<thead>
<tr>
<th># of Memes</th>
<th># of Domains</th>
<th>Density</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>21</td>
<td>98.2%</td>
<td>St. Jude’s Children’s Hospital</td>
</tr>
<tr>
<td>178</td>
<td>5</td>
<td>96.1%</td>
<td>Brazilian news sites</td>
</tr>
<tr>
<td>39</td>
<td>6</td>
<td>98.7%</td>
<td>Spanish news sites</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>99.2%</td>
<td>Tech conference and magazine sites</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>100.0%</td>
<td>Political blogs</td>
</tr>
</tbody>
</table>

Table 2: Top biclusters returned by the CC-MACS algorithm on the Meme-Tracker dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original Matrix</th>
<th>Cross-Association</th>
<th>CC-MACS ($\mu_0$)</th>
<th>CC-MACS ($\mu_1$)</th>
<th>CC-MACS ($\mu_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIDAP005</td>
<td><img src="image1" alt="Matrix Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
<td><img src="image5" alt="Image" /></td>
</tr>
<tr>
<td>FIDAPM05</td>
<td><img src="image6" alt="Matrix Image" /></td>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
</tr>
<tr>
<td>QC324</td>
<td><img src="image11" alt="Matrix Image" /></td>
<td><img src="image12" alt="Image" /></td>
<td><img src="image13" alt="Image" /></td>
<td><img src="image14" alt="Image" /></td>
<td><img src="image15" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 9: Real-world datasets from finite element modeling (FIDAP005 and FIDAPM05) and quantum chemistry (QC324). The red lines indicate co-clusterings found by Cross-Association and CC-MACS algorithms.


