Improving Co-Cluster Quality with Application to Product Recommendations

Michail Vlachos [*]
IBM Research, Zurich

Francesco Fusco
IBM Research, Zurich

Charalambos Mavroforakis
Boston University

Anastasios Kyrillidis
EPFL, Lausanne

Vassilios G. Vassiliadis
IBM Research, Zurich

ABSTRACT

Businesses store an ever increasing amount of historical customer sales data. Given the availability of such information, it is advantageous to analyze past sales, both for revealing dominant buying patterns, and for providing more targeted recommendations to clients. In this context, co-clustering has proved to be an important data-modeling primitive for revealing latent connections between two sets of entities, such as customers and products.

In this work, we introduce a new algorithm for co-clustering that is both scalable and highly resilient to noise. Our method is inspired by k-Means and agglomerative hierarchical clustering approaches: (i) first it searches for elementary co-clustering structures and (ii) then combines them into a better, more compact, solution. The algorithm is flexible as it does not require an explicit number of co-clusters as input, and is directly applicable on large data graphs. We apply our methodology on real sales data to analyze and visualize the connections between clients and products. We showcase a real deployment of the system, and how it has been used for driving a recommendation engine. Finally, we demonstrate that the new methodology can discover co-clusters of better quality and relevance than state-of-the-art co-clustering techniques.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering

1. INTRODUCTION

Graphs are popular data abstractions, used for compact representation of datasets and for modeling connections between entities. When studying the relationship between two classes of objects (e.g., customers vs. products, viewers vs. movies, etc.), *bipartite graphs*, in which every edge in the graph highlights a connection between objects in different classes, arise as a natural choice for data representation. Owing to their ubiquity, bipartite graphs have been the focus of a broad spectrum of studies, spanning from document analysis [7] and social-network analysis [4] to bioinformatics [14] and biological networks [16]. Here we focus on business intelligence data, where a bipartite graph paradigm represents the buying pattern between sets of customers and sets of products. Analysis of such data is of great importance for businesses, which accumulate an ever increasing amount of customer interaction data.

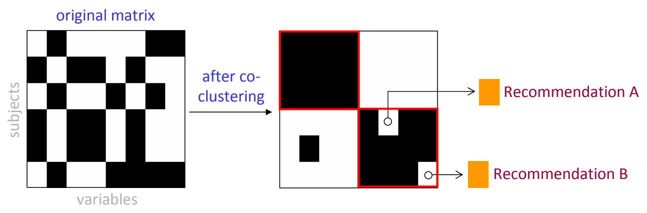

Figure 1: Matrix co-clustering can reveal the latent structure. Discovered 'white spots' within a co-cluster can be coupled with a recommendation process.

One common process in business data intelligence is the identification of groups of customers who buy (or do not buy) a subset of products. Such information is advantageous to both the sales and marketing teams: Sales people can exploit these insights to offer more personalized (and thus more accurate) product suggestions to customers by examining the behavior of "similar" customers. At the same time, identification of buying/not-buying preferences can assist marketing people in determining groups of customers interested in a subset of products. This, in turn, can help orchestrate more focused marketing campaigns, and lead to more judicious allocation of the marketing resources.

In our context, we are interested to understand the connections between customers and products. We represent the buying patterns as binary matrix. The presence of black square (a 'one') signifies that a customer has bought a product, otherwise the square is white ('zero'). Given such a matrix data representation, the problem of discovering sets of correlated sets of customers and products can be cast as a *co-clustering problem instance* [1, 6, 11]. Such a process will result in a permutation of rows and columns, such that the resulting matrix is as homogeneous as possible. It will also reveal any latent group structure of a seemingly unstructured original matrix. Figure 1 shows an example of the original and the co-clustered matrix, where the rows represent customers and the columns products.

It is apparent that the reordered matrix (Figure 1, right) provides strong evidence on the presence of buying patterns. Moreover, we can use the discovered co-clusters to provide targeted *product recommendations* to customers as follows: *'white spots'* within a co-cluster suggest potential product recommendations. These recommendations can further be ranked based on firmographic informa-

[*]The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 259569.

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright 2014 ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2661980 .

tion of the customers (revenue, market growth, etc.).

Well-established techniques for matrix co-clustering have been based on: hierarchical clustering [9], centroid-based clustering (e.g., k-Means based), or spectral clustering principles of the input matrix [7]. As we discuss in more detail later on, each of these approaches individually can exhibit limited scalability, poor recovery of the true underlying clusters, or reduced noise resilience. In this work, we present a hybrid technique that is both *scalable*, supporting the analysis of thousands of graph nodes, and *accurate* in recovering many cluster structures that previous approaches fail to distinguish. Our **contributions** are:

- We provide a new scalable solution for co-clustering binary data. Our methodology consists of two steps: (i) an initial seeding and fast clustering step, (ii) followed by a more expensive refinement step, which operates on a much smaller scale than the ambient dimension of the problem. Our approach showcases *linear time-cost* and *space-complexity* with respect to the matrix size. More importantly, it is extremely noise-resilient, and easy to implement.
- In practice, the true number of co-clusters is not known *a-priori*. Thus, an inherent limitation of many co-clustering approaches is the explicit specification of the parameter K - the number of clusters per dimension.[1] Our method is more flexible, as it only accepts as input a rough upper estimate on the number of co-clusters. Then it explores the search space for more compact co-clusters, and the process terminates automatically when it detects an anomaly in the observed entropy of the compacted matrix.
- We leverage our co-clustering solution as the foundation in a B2B (Business to Business) recommender system. The recommendations are ranked using both *global* patterns, as discovered by the co-clustering procedure, and *personalized* metrics, attributed to each customer's individual characteristics.

To illustrate the merits of our approach, we perform a comprehensive empirical study on both synthetic and real data to validate the quality of solution, as well the scalability of our approach, and compare it with state-of-the-art co-clustering techniques.

Paper organization: We start in Section 2 by reviewing the related work. Section 3 describes the overall problem setting, gives an overview of the proposed co-clustering methodology, and explains how it can be incorporated within a product recommendation system. Section 4 describes our co-clustering technique in detail. Section 5 evaluates our approach and compares it with other co-clustering techniques. Finally, Section 6 concludes our description and examines possible directions for future work.

2. RELATED WORK

The principle of co-clustering was first introduced by Hartigan with the goal of 'clustering cases and variables simultaneously' [11]. Initial applications were for the analysis of voting data. Since then, several co-clustering algorithms have been proposed, broadly belonging to four classes: a) hierarchical co-clustering, b) spectral co-clustering, c) information-theoretic co-clustering, and d) optimization-based co-clustering.

Hierarchical co-clustering: these approaches are typically the choice of preference in biological and medical sciences [14, 18]. In these disciplines, co-clustering appears under the term 'bi-clustering'. For an example see Fig. 2. Agglomerative hierarchical co-clustering

[1] In most test cases, the number of clusters per dimension is not equal. To be precise, we use K and L to denote the number of clusters for each dimension. For clarity, we keep only K in our discussions, unless stated otherwise.

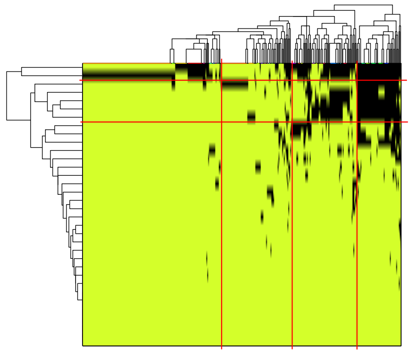
Figure 2: Agglomerative Hierarchical co-clustering

approaches can lead to the discovery of very compact clusters and are parameter-free; a fully extended tree is computed and the user can decide interactively on the number of co-clusters (i.e., where the tree is 'cut'). Despite the high quality of derived co-clusters, hierarchical clustering approaches come with an increased runtime cost: it ranges from $O(n^2)$ to $O(n^2 \log^2 n)$ depending on the agglomeration process [10], n being the number of objects. In the general case, the time complexity is $O(n^3)$. Therefore, their applicability is limited to data with few hundreds of objects and is deemed prohibitive for big data instances.

Spectral co-clustering: here, the co-clustering problem is solved as an instance of graph partitioning (k-cut) and can be relegated to an eigenvector computation problem [7]. These approaches are powerful as they are invariant to cluster shapes and densities (e.g., partitioning $2D$ concentric circles). Their computational complexity is dominated by the eigenvector computation: in the worst-case scenario, this computation has cubic time complexity; in the case of sparse binary co-clustering, efficient iterative Krylov and Lanczos methods can be used with $O(n^2)$ complexity.[2] However, in our case, one is interested in detecting rectangular clusters; hence, computationally simpler techniques show similar or even better clustering performance. Recent implementations report a runtime of several seconds for a few thousands of objects [15]. As k-Means is usually inherent in such approaches, an estimate on the number of clusters should be known a-priori; thus, in stark contrast to hierarchical co-clustering, spectral algorithms are re-executed for each different K value. Spectral-based clustering techniques can recover high-quality co-clusters in the absence of noise, but their performance typically deteriorates for noisy data. They may also introduce spurious co-clusters, when the data consists of clusters with very different sizes. For visual examples of these cases, see Figure 3.

Information-theoretic co-clustering: this thrust of algorithms is based on the work of Dhillon et al. [8]. Here, the optimal co-clustering solution maximizes the mutual information between the clustered random variables and results into a $K \times K$ clustered matrix, where K is user-defined. Crucial for its performance is the estimation of the joint distribution $p(X,Y)$ of variables and subjects; in real-world datasets, such an estimate is difficult (if not impossible) to compute with high accuracy. According to the original authors, the resulting algorithm has $O(\mathrm{nz} \cdot \tau \cdot K)$ time cost [8], where nz is the number of non-zeros in the input joint distribution $p(X,Y)$ and τ is the total number of iterations to converge. Only

[2] We should highlight that while the *eigenvalue* computation using these methods has a well-studied complexity, the corresponding *exact eigenvector* (up to numerical accuracy) can be computationally hard to estimate [12]. A variant of the Lanczos method with random starting vector, where only probabilistic approximation guarantees are given, is proposed in [2].

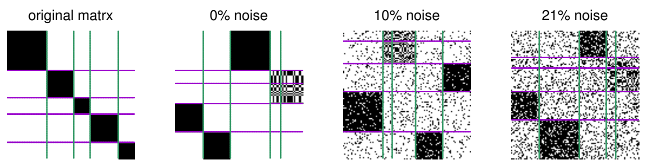

Figure 3: Spectral co-clustering using the Fiedler vector. We can observe that it cannot recover the existing co-clusters accurately, even in the absence of noise.

empirical insights on the upper bound for τ have been provided.

Optimization-based co-clustering: these methodologies use various optimization criteria to solve the co-clustering problem. Typical choices may include information-theoretic-based objective functions [17], or other residue functions [5]. The computational complexity is on order of $O(n^2)$.

3. PROBLEM SETTING

Assume a bipartite graph of customers versus products, where the existence of an edge indicates that a customer has bought a particular product. The information recorded in the graph can also be conveyed in an adjacency matrix, as shown in Figure 4. The adjacency matrix contains the value of 'one' at position (i, j) if there exists an edge between the nodes i and j; otherwise the value is set to 'zero'. Note that the use of the matrix representation also enables a more effective visualization of large graph instances.

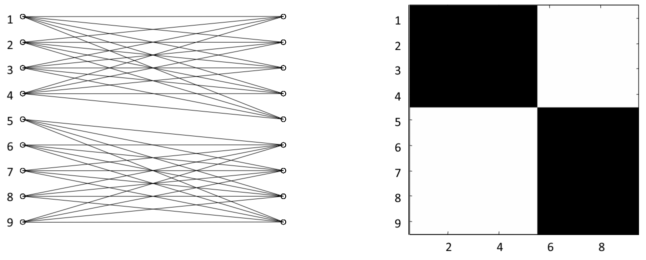

Figure 4: Left: Bipartite graph representation. Right: Adjacency matrix representation.

Initially, this adjacency matrix has no orderly format: typically, the order of rows and columns is random. Our goal is to extract any latent cluster structure from the matrix and use this information to recommend products to customers. We perform the following actions, as shown in Figure 5:

1. First, we reorganize the matrix to reveal any hidden co-clusters in the data.
2. Given the recovered co-clusters, we extract the 'white spots' in the co-clusters as potential recommendations.
3. We rank these recommendations from stronger to weaker, based on available customer information.

4. CO-CLUSTERING ALGORITHM

Our methodology accomplishes a very compact co-clustering of the adjacency matrix. We achieve this by following a two-step approach: the initial fast phase (`Cluster` phase) *coarsens* the matrix and extracts elementary co-cluster pieces. A second phase (`Merge` phase) iteratively *refines* the discovered co-clusters by progressively merging them. The second phase can be perceived as *piecing together the parts of a bigger puzzle*, as we try to identify which pieces (co-clusters) look similar and should be placed adjacent to each other.

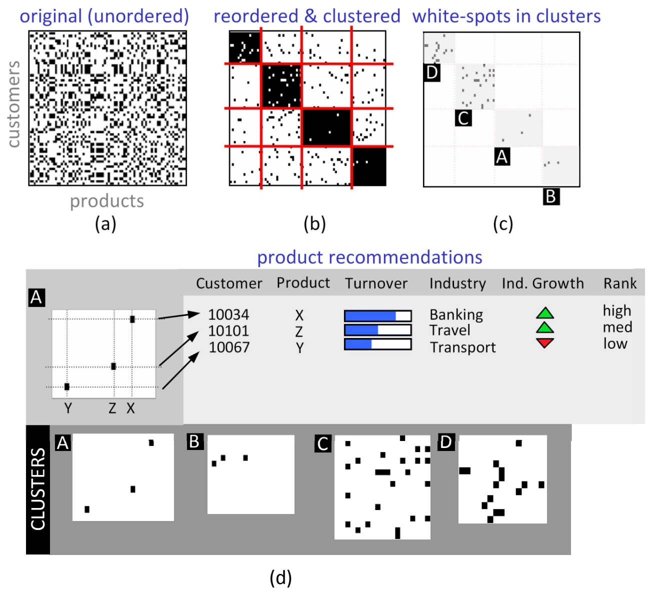

Figure 5: Overview of our approach: a) Original matrix of customers-products, b) matrix co-clustering, c) 'white spots' within clusters are extracted, d) product recommendations are identified by ranking the white spots based on known and prognosticated firmographic information.

In practice, one can visualize the whole strategy as a hybrid approach, in which a double k-Means initialization is followed by an agglomerative hierarchical clustering. As we show in more detail in subsequent sections, the above process results in a co-clustering algorithm that is extremely robust to noise, exhibits linear scalability as a function of the matrix size, and recovers very high quality co-clusters.

To determine when the algorithm should stop merging the various co-cluster pieces, we use entropy-based criteria. However, the presence of noise may lead to many local minima in the entropy. We avoid them by looking for *large deviants* in the entropy measurements. So, we model the stopping process as an anomaly detector in the entropy space. The end result is an approach that does not require a fixed number of co-clusters, but only a rough estimate for the upper bound of co-clusters, i.e., the number of clusters given to the k-Means cluster step. From then on, it searches and finds an appropriate number of more compact co-clusters. Because we model the whole process as a detection of Entro**Py** **A**nomalies in **Co**-Clustering, we call the algorithm `PaCo` for short. A visual illustration of the methodology is given in Figure 6.

4.1 The `PaCo` Algorithm

Assume an *unordered* binary matrix $\boldsymbol{X} \in \{0,1\}^{N \times M}$ which we wish to co-cluster along both dimensions. Row clustering treats each object as a $\{0,1\}^M$ vector. Similarly, column clustering considers each object as a $\{0,1\}^N$ vector derived by transposing each column. We use K and L to denote the number of clusters in rows and columns of $\boldsymbol{X}$, respectively.

`Cluster` Phase: To extract elementary co-cluster structures from $\boldsymbol{X}$, we initially perform *independent* clustering on rows and columns. Then, we combine the discovered clusters per dimension to form the initial co-clusters, which we will use in the `Merge` phase. To

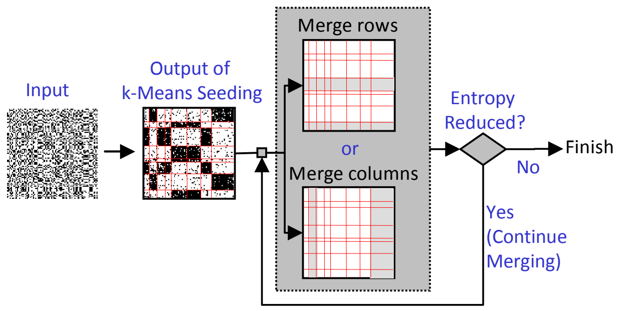

Figure 6: Overview of the proposed co-clustering process. *k*-Means clustering is performed on both rows and columns and subsequently closest block rows and columns are merged together. Entropy-based stopping criterion based on past merging operations: as long as the entropy does not deviate from the average, the merging process continues.

achieve the above, we use a centroid-based *k*-Means algorithm per dimension. To increase its efficiency, we choose the initial centroids according to the *k*-Means++ variant [3]: this strategy generates centroid seeds that lead to provably good initial points, and has been shown to be very stable and within bounded regions with respect to the optimal solution. Moreover, recent work in approximation theory has shown that performing *k*-Means separately on each dimension provides constant factor approximations to the best co-clustering solution under a *k*-Means-driven optimization function [1]. Therefore, we expect the outcome of the `Cluster` phase to reside within rigid quality bounds from the optimal solution.

This phase results in a $K \times L$ block matrix. Note, that we don't need to explicitly indicate the number of final co-clusters. The values K and L that we provide in this phase are only *rought, upper bounds estimates* on the true number of clusters $K^\star$ and $L^\star$. From there on, the subsequent phase tries to merge the resulting co-clusters. As an example, in our experiments, we use $K = L = 50$, because we only expect to finally display 5 to 20 co-clusters to the end user. Note, however, that the actual values of this initial coarse co-clustering phase do not directly affect the quality but rather the runtime. We show this later in this analysis of the algorithm complexity.

`Merge` Phase: We start the second phase having a $K \times L$ block matrix. Now, the second phase gets initiated, a process of *moving* blocks of co-clusters such that the rearrangement results in a more *homogeneous* and *structured* matrix.

Therefore, in this phase we try to identify similar rows or columns of co-clusters which can be merged. Before we define our similarity measure for co-cluster blocks, we explain some basic notions. For every cluster i in the j-th row (column resp.) of the $K \times L$ block matrix, let $s_j(i)$ ($s^j(i)$ resp.) denote the number of cells it contains and we use the notation $\mathbb{1}_j(i)$ ($\mathbb{1}^j(i)$ resp.) to represent the total number of nonempty cells ('ones') in the cluster i. Then, the density of this cluster is defined as $d_j(i) = \frac{\mathbb{1}_j(i)}{s_j(i)}$ (and thus $d^j(i)$ resp.).[3] We easily observe that $d_j(i) \to 1$ denotes a dense cluster. whereas $d_j(i) \to 0$ denotes an empty cluster.

Given this definition, to assess the similarity between the p-th and q-th rows (columns resp.) in the $K \times L$ matrix, we treat each block row as vectors

$$\mathbf{v}_p = (d_p(1) \;\; d_p(2) \;\; \ldots \;\; d_p(K))^T$$

[3]Without loss of generality and for clarity, we might use $d(i)$ to denote a co-cluster in the matrix, without specifying the block row/column it belongs to.

and

$$\mathbf{v}_q = (d_q(1) \;\; d_q(2) \;\; \ldots \;\; d_q(K))^T,$$

with entries equal to the densities of the corresponding clusters — we can similarly define $\mathbf{v}^p$ and $\mathbf{v}^q$, but, for the sake of clarity, we will only focus on the row case. A natural choice to measure the distance between two vectors is the Euclidean distance: their distance in the ℓ_2-norm sense is given as

$$D(\mathbf{v}_p,\, \mathbf{v}_q) = \frac{\|\mathbf{v}_p - \mathbf{v}_q\|_2^2}{K} \tag{1}$$

The density vectors are normalized by their length, because the merging process may result in different number of rows or column blocks and, therefore, it is necessary to compensate for this discrepancy (i.e., when examining whether to merge rows or columns). Then, the merging pair of rows is given by

$$\{p^\star,\, q^\star\} \in \arg\min_{p,q \in \{1,\ldots,K\}, p \neq q} D(\mathbf{v}_p,\, \mathbf{v}_q), \tag{2}$$

where any ties are dissolved lexicographically. Figure 7 shows two iterations of the merging process. In step r, columns 4 and 1 are merged as the most similar (smallest distance) of all pairs of columns/rows. At step $r+1$, rows 6 and 2 are chosen for merging, because now they are the most similar, and so on.

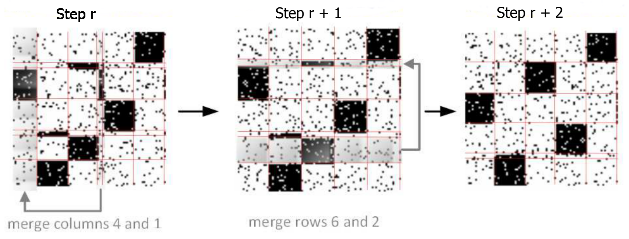

Figure 7: Two iterations of the algorithm.

Stopping criterion: We evaluate when the merging process should terminate by adapting an information-theoretic criterion.

Definition 1 (Entropy measure). *Consider a set of positive real numbers $P = \{p_1, p_2, \ldots, p_n\}$ such that $\sum_{i=1}^n p_i = 1$. The entropy is defined as $H(P) = -\sum_{i=1}^n p_i \log p_i$. Since $H(P) \in [0, \log n]$ for every set of size n, we compare entropy values of different-sized sets normalizing accordingly: $H_n(P) = \frac{H(P)}{\log n} \in [0,1]$.*

Entropy measures how uneven a distribution is. In our setting, it assesses the distribution of the recovered non-empty dense co-clusters in the matrix. By normalizing the densities by $d_{\text{sum}} = \sum_{i=1}^{KL} d(i)$, we can compute the entropy of the set of normalized densities $p_i = \frac{d(i)}{d_{\text{sum}}}$.

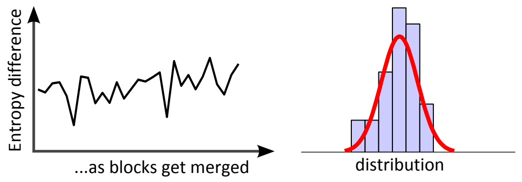

Figure 8: The differences in the entropy value can be modeled as a Gaussian distribution.

As similar co-clusters are merged, the entropy of the matrix is reduced. However, because noise is typically present, the first increase in the entropy does not necessarily suggest that the merging process should terminate. To make the process more robust,

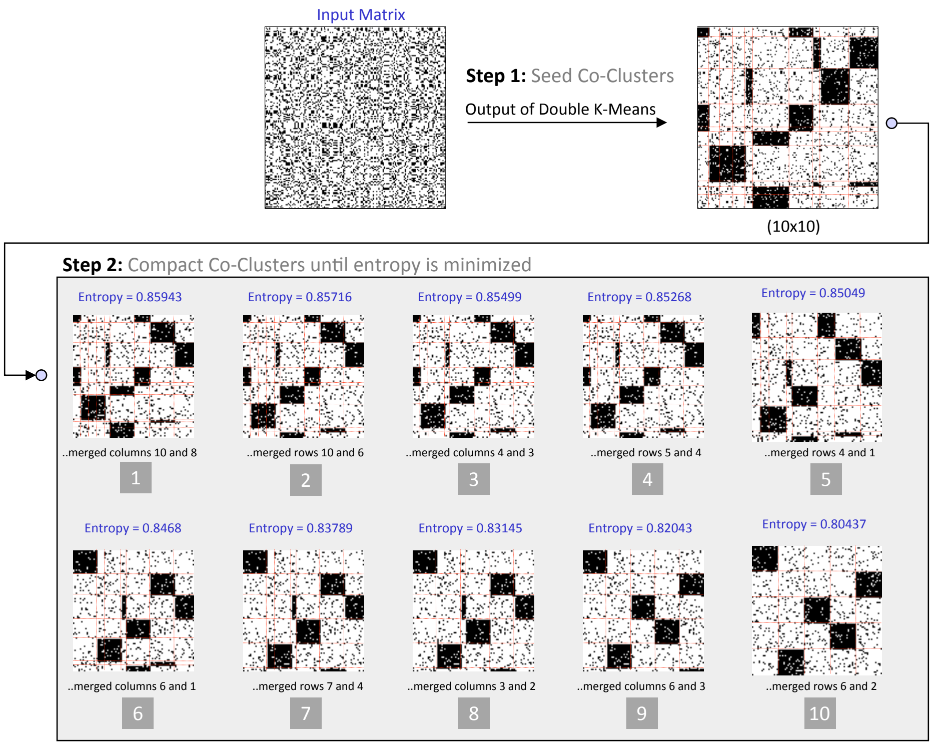

Figure 9: A sample run of our algorithm. First, the rows and columns of the matrix are clustered. Then, as long as there is no anomaly observed in the entropy difference, a pair of either block rows or columns is merged.

the algorithm monitors the history of entropy values for the matrix. We observe that the entropy differences from one matrix state to the subsequent one follows a highly Gaussian distribution. An instance of this for real-world data is depicted in Figure 8. Therefore, we will terminate the merging process when a *large anomaly* is observed in the matrix entropy, e.g. outside 3 standard deviations from the observed history of entropy differences. This allows the process to be particularly robust to noise and to discover the appropriate stable state of the system.

Example: Figure 10 shows a non-fictional example of the stopping criterion. Small differences in entropy (either up or down) do not terminate the merging. However, merging the 5×5 block state of the system into 5×4 blocks introduces a very large anomaly. Here the merging terminates, and the final state of the system will be with a 5×5 block of co-clusters.

A pseudocode of the whole process described so far is provided in Algorithm 1. Also, an actual run of the algorithm is visually demonstrated in Figure 9. The figure shows all the inbetween merge steps leading from a 10×10 block to a 5×5 block of co-clusters.

Complexity: The `Cluster` phase involves two independent k-Means operations. The time complexity is $O(M \cdot N \cdot \max\{K,L\} \cdot I)$, where I is the number of iterations taken by the k-Means algorithm to converge. As K,L,I are constant and fixed in advanced, the time complexity is *linear* in the size of the data set. In practice, the expected complexity for k-Means clustering is significantly lower because we deal with very sparse matrices. In this case, the time cost is $O(\text{nnz}(\boldsymbol{X}) \cdot \max\{K,L\} \cdot I)$, where $\text{nnz}(\boldsymbol{X})$ is the number of non-zero elements in the matrix. The space complexity for this phase is upper-bounded by $O(MN + \max\{K,L\})$ to store the matrix and some additional information.

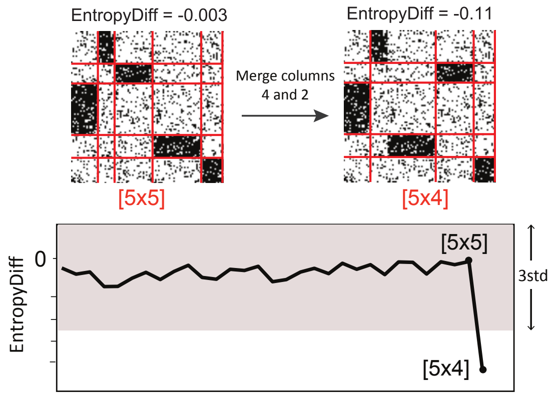

Figure 10: To stop the merging process we look for deviants in the entropy distribution.

During the `Merge` phase, blocks of rows or blocks of columns are merged as long as the stopping criterion is not violated; thus, there can be at most $K+L$ iterations. At every iteration, Steps 6 and 7 are calculated in $O(KL)$ time cost and with $O(KL)$ space complexity. Steps 8 and 9 require the computation of $\binom{K}{2}$ block row distances ($\binom{L}{2}$ block column distances resp.), with $O(K)$ time cost for each distance computation. The space complexity is $O(KL)$. The merging operation in Step 13 can be computed in $O(1)$ time. As the number of clusters per dimension decreases per iteration (depending on whether we merge w.r.t. rows or columns), we observe that the total cost over all iterations is at most $O(\max\{K,L\}^4)$.

Overall, the algorithm has $O(M \cdot N \cdot \max\{K,L\} \cdot I + \max\{K,L\}^4)$

Algorithm 1 The `PaCo` co-clustering algorithm

```
 1: procedure  $\{\widehat{\boldsymbol{X}}, \texttt{R}, \texttt{C}\} = \texttt{PaCo}(\boldsymbol{X}, \text{K}, \text{L})$        ▷ $\boldsymbol{X} \in \{0,1\}^{N\times M}$
                              Cluster phase
 2:   $\texttt{R} = \{R_1, R_2, \ldots, R_K\} \leftarrow k$-Means++(set of rows of $\boldsymbol{X}$, $K$)
 3:   $\texttt{C} = \{C_1, C_2, \ldots, C_L\} \leftarrow k$-Means++(set of columns of $\boldsymbol{X}$, $L$)
 4:   $\widehat{\boldsymbol{X}} \leftarrow$ REARRANGE($\boldsymbol{X}$, R, C)
                              Merge phase
 5:   while Stopping criterion is not met do
 6:       Compute the density matrix $\mathbf{V} \in \mathbb{R}^{K\times L}$.
 7:       $\mathbf{V}(\mathbf{V} < \texttt{density\_low}) = 0$.        ▷ Ignore "sparse" clusters
 8:       $\{\texttt{mergeR}, R_i, R_j\} \leftarrow$ CHECK($\mathbf{V}$, R)
 9:       $\{\texttt{mergeC}, C_g, C_h\} \leftarrow$ CHECK($\mathbf{V}$, C)
10:       If ($\texttt{mergeR} == \texttt{mergeC} == \texttt{False}$): break
11:       else
12:           $\{T_1, T_2\} = \arg\max\big\{\{R_i, R_j\}, \{C_g, C_h\}\big\}$ ▷ Pick most similar pair
13:           Merge the clusters in $\{T_1, T_2\}$ and update $\widehat{\boldsymbol{X}}$ and $\texttt{R}, K$ (or $\texttt{C}, L$).
14:       end if
15:   end while
16: function  $\{\texttt{merge}, T_i, T_j\} = $ CHECK($\mathbf{V}$, T)
17:   Compute row/column distances $\|\mathbf{v}_p - \mathbf{v}_q\|_2^2$, $\forall p, q \in \{1, \ldots, |\texttt{T}|\}$.
18:   Pick $p, q$ with the min. distance s.t. the merged block has high
      enough density-per-cocluster (e.g., $\geq \texttt{density\_high}$) and the en-
      tropy increase does not deviate from the mean.
19:   If no such pair exists: return $\{\texttt{False}, [], []\}$
20:   else return $\{\texttt{true}, p, q\}$
```

time cost and $O(KL + MN + \max\{K, L\})$ space complexity. Note that K, L is the number of initial clusters in rows and columns respectively, which are constant and usually small; hence, in practice, our algorithm exhibits linear runtime with respect to the matrix size.

4.2 Parallelizing the process

The analysis suggested that the computationally more demanding portion is attributed to the k-Means part of the algorithm; the merging steps are only a small constant fraction of the whole cost.

To explore further runtime optimizations of our algorithm, we also implemented a parallel-friendly version of the first phase. In this way the algorithm fully exploits the multi-threading capabilities of modern CPUs. Instead of a single thread updating the centroids and finding the closest centroid per point in the k-Means computation, we assign parts of the matrix to different threads. Now, when a computation involves a particular row or column of the matrix, the computation is assigned to the appropriate thread.

Our parallel implementation of k-Means uses two consecutive parallel steps: (i) in the first step, called `updateCentroids`, we compute new centroids for the given dataset in parallel. (ii) In the second step, called `pointReassign`, we re-assign each point to the centroids computed in the preceding step. Both steps work by equally partitioning the dataset between threads. A high-level pseudocode is provided in Algorithm 2.

In the experiments we show the above simple extension parallelizes the co-clustering process with high efficiency. Note, that for parallelization we didn't use a Hadoop implementation, because Hadoop is primarily intended for big but offline jobs, whereas we are interested in (almost) real-time execution.

5. EXPERIMENTS

We illustrate the ability of our algorithm to discover patterns hidden in the data. We compare it with state-of-the-art co-clustering approaches and show that our methodology is able to recover the underlying cluster structure with greater accuracy. We also demonstrate a prototype of our co-clustering algorithm coupled with a recommendation engine within a real industrial application. We also compare the recommendation power of co-clustering with the recommendations derived via traditional techniques based on association rules.

Algorithm 2 Parallelization of `PaCo` initialization

```
 1: function updateCentroids($\boldsymbol{X}, T$)                ▷ $T$: number of threads
 2:   Partition rows/columns of $\boldsymbol{X}$ into $P_1, P_2, \ldots, P_T$ with cardinality $|P_i| =$
      $M/T$ or $|P_i| = N/T$, resp.
 3:   for each thread $t$ in $T$ do
 4:     Compute $K(L)$ centroids $C = c_1^{(t)}, c_2^{(t)}, \ldots, c_K^{(t)}$ (or $c_L^{(t)}$) using $P_t$.
 5:   end end
 6:     Compute new centroids by summing and averaging $C =$
      $c_1^{(t)}, c_2^{(t)}, \ldots, c_K^{(t)}$.
 7: function pointReassign($\boldsymbol{X}, C$)
 8:   Partition rows/columns of $\boldsymbol{X}$ into $P_1, P_2, \ldots, P_T$ with cardinality $|P_i| =$
      $M/T$ or $|P_i| = N/T$, resp.
 9:   for each thread $t$ in $T$ do
10:     Finds nearest centroid in C for each row (column resp.) in $P_t$.
11:   end end
12:   Reassign data rows (columns) to centroids.
```

Algorithms: We compare the `PaCo` algorithm with two state-of-the-art co-clustering approaches: (i) an Information-Theoretic Co-Clustering algorithm (INF-THEORETIC) [8] and, (ii) a Minimum Sum-Squared Residue Co-Clustering algorithm (MSSRCC) [5]. We use the original and publicly available implementations, provided by the original authors.[4]

Co-Cluster Detection Metric: When the ground truth for co-clusters is known, we evaluate quality of co-clustering using the notion of *weighted co-cluster relevance* $R(\cdot,\cdot)$ [5]:

$$R(\texttt{M}, \texttt{M}_{\text{opt}}) = \frac{1}{|\texttt{M}|} \sum_{(\texttt{R},\texttt{C})\in\texttt{M}} \frac{|\texttt{R}||\texttt{C}|}{|\texttt{R}_{\text{total}}||\texttt{C}_{\text{total}}|} \cdot \max_{(\texttt{R}_{\text{opt}},\texttt{C}_{\text{opt}})\in\texttt{M}_{\text{opt}}} \left\{ \frac{|\texttt{R}\cap\texttt{R}_{\text{opt}}|}{|\texttt{R}\cup\texttt{R}_{\text{opt}}|} \frac{|\texttt{C}\cap\texttt{C}_{\text{opt}}|}{|\texttt{C}\cup\texttt{C}_{\text{opt}}|} \right\}.$$

Here, $\texttt{M}$ is the set of co-clusters discovered by an algorithm and $\texttt{M}_{\text{opt}}$ are the true co-clusters. Each co-cluster is composed of a set of rows $\texttt{R}$ and columns $\texttt{C}$.

5.1 Accuracy, Robustness and Scalability

First we evaluate the accuracy and scalability across co-clustering techniques by generating large synthetic datasets, where the ground-truth is known. To generate the data, we commence with binary, block-diagonal matrices (where the blocks have variable size) that simulate sets of customers buying sets of products and distort them using 'salt and pepper' noise. Addition of noise p means that the value of every entry in the matrix is inverted (0 becomes 1 and vice versa) with probability p. The rows and columns of the noisy matrix are shuffled, and this is the input to each algorithm.

Co-cluster Detection: Table 1 shows the co-cluster relevance of our approach compared with the Minimum Sum-Squared Residue (MSSRCC) and the Information-Theoretic approaches. For this experiment we generated matrices of increasing sizes ($10{,}000 - 100{,}000$ rows). Noise was added with probability $p = 0.2$. The reported relevance $R(\cdot,\cdot)$ corresponds to the average relevance across all matrix sizes. We observe that our methodology recovers almost

[4] `http://www.cs.utexas.edu/users/dml/Software/cocluster.html`

all of the original structure, and improves the co-cluster relevance of the other techniques by as much as 60%.

Table 1: Co-cluster relevance of different techniques. Values closer to 1 indicate better recovery of the original co-cluster structure.

	Relevance $R(\cdot,\cdot)$	Relative Improvement
PaCo	0.99	-
MSSRCC	0.69	43%
Inf-Theoretic	0.62	60%

Resilience to Noise: Real-world data are typically noisy and do not contain clearly defined clusters and co-clusters. Therefore, it is important for an algorithm to be robust, even in the presence of noise. In Figure 11, we provide one visual example that attests to the noise resilience of our technique. Note that our algorithm can accurately detect the original patterns, without knowing the true number of co-clusters in the data. In contrast, we explicitly provide the *true* number $K^\star = L^\star = 5$ of co-clusters to the techniques under comparison. Still, we note that in the presence of noise ($p = 15\%$), the other methods return results of lower quality.[5]

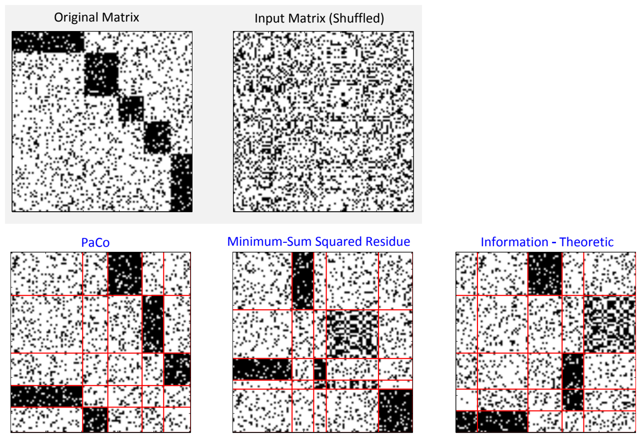

Figure 11: Co-clustering synthetic data example in the presence of noise. The input is a shuffled matrix containing a 5×5 block pattern. Our approach does not require as input the true number of co-clusters $K^\star, L^\star$, as it automatically detects the number of co-clusters; here, we set $K = L = 10$. In contrast, for the competitive techniques, we provide $K = K^\star$ and $L = L^\star$. Still, the structure they recovered is of inferior quality.

Scalability: We evaluate the time complexity of PaCo in comparison to the MSSRCC and the Information-Theoretic co-clustering algorithms. All algorithms are implemented in C++ and executed on an Intel Xeon at 2.13Ghz.

The results for matrices of increasing size are shown in Figure 12. The runtime of PaCo is equivalent or lower than other co-clustering approaches. *Therefore, even though it can recover co-clusters of significantly better quality (relative improvement in quality $40-60\%$), this does not come at the expense of extended runtime.*

Parallelization in PaCo: We achieved the previous runtime results by running the PaCo algorithm on a single system thread. As discussed, the process can easily be parallelized. Here, we evaluate

[5]In the figure, the order of the discovered co-clusters is different from that in the original matrix. The output can easily be standardized to the original block-diagonal, by an appropriate reordering of the co-cluster outcome.

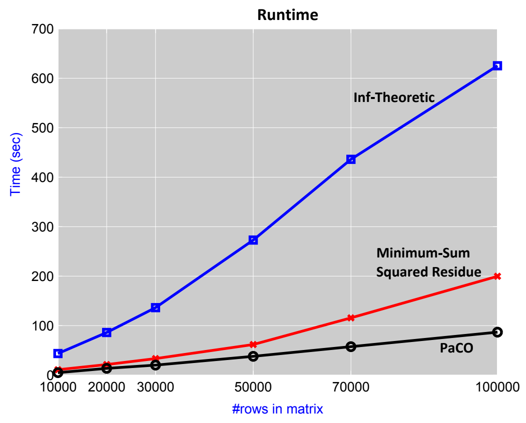

Figure 12: Scalability of co-clustering techniques. Notice the linear scalability of our approach.

how much further the co-clustering process can be sped up, using a single CPU, but now exploiting the full multi-threading capability of our system.

The computational speedup is shown in Figure 13 for the case of at most $T = 8$ threads. We see the merits of parallelization; we gain up to $\times5.1$ speedup, without needing to migrate to a bigger system.

It is important to observe that after the 4th thread, the efficiency is reduced. This is the case because we used a 4-core CPU with Hyper-Threading (that is, 8 logical cores). Therefore, scalability after 4 cores is lower because for threads 5,6,7,8 we are not actually using physical cores but merely logical ones. Still, the red regression line suggests that the problem is highly parallel (efficiency ~ 0.8), and on a true multi-core system we can fully exploit the capabilities of modern CPU architectures.

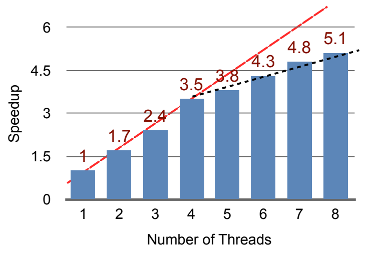

Figure 13: Speedup achieved using the parallel version of PaCo. Note that the reduction in performance after the 4th thread was due to the fact that we used 4-core CPU with HyperThreading (8 logical cores).

5.2 Enterprise deployment

We examined the merits of our algorithm in terms of robustness, accuracy and scalability. Now, we describe how we used the algorithm on a real enterprise environment. We capitalize on the co-clustering process as basis for: a) understanding the buying patterns of customers and, b) forming recommendations, which are then forwarded to the sales people responsible for these customers.

In a client-product matrix, white areas inside co-clusters represent clients that exhibit similar buying patterns as a number of other clients, but still have not bought some products within their respective co-cluster. These white spots represent products that constitute good recommendations. Essentially, we exploit the existence of *globally-observable* patterns for making individual recommendations.

However, not all white spots are equally important. We rank

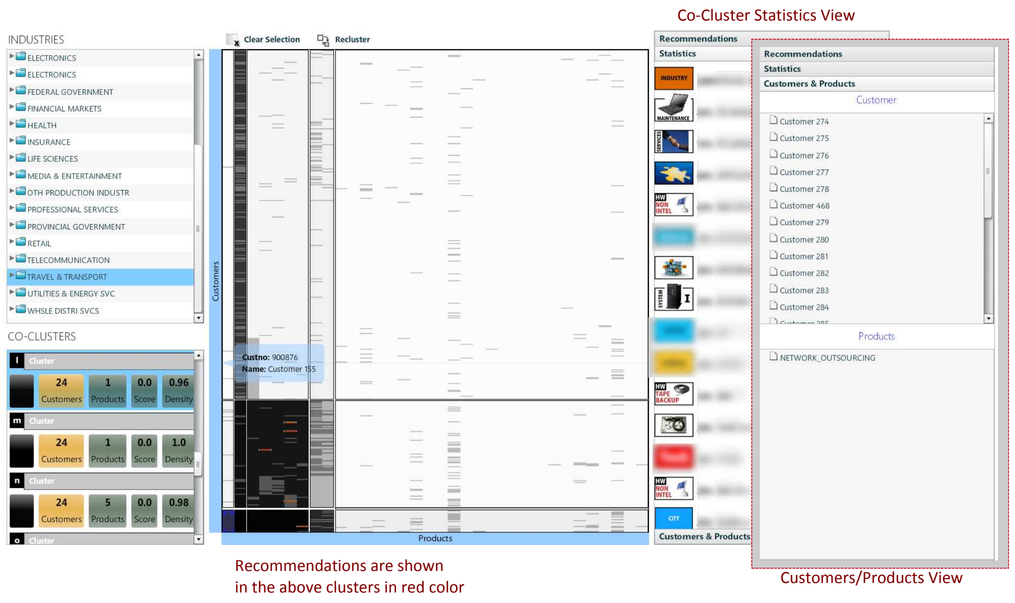

Figure 14: Graphical interface for the exploration of client-product co-clusters

them by considering firmographic and financial characteristics of the clients. In our scenario, that the clients are not individuals, but large companies for which we have extended information, such as: the *industry* to which they belong (banking, travel, automotive, etc), *turnover* of the company/client, past buying patterns etc. We use all this information to rank the recommendations. The intuition is that 'wealthy' clients/companies that have bought many products in the past are better-suited candidates. They have the financial prowess to buy a product and there exists an already established buying relationship. Our ranking formula considers three factors:

- Turnover,TN, the revenue of the company as provided in its financial statements.
- Past Revenue, RV, the amount of financial transactions our company had in its interactions with the customer in the past 3 years.
- Industry Growth, IG, the predicted growth for the upcoming year for the industry (e.g. banking, automotive, travel,...) to which the customer belongs. This data is derived by marketing databases and is estimated from diverse global financial indicators.

The final rank r of a given white spot that captures a customer-product recommendation is given by:

$$r = w_1\mathrm{TN} + w_2\mathrm{RV} + w_3\mathrm{IG}, \quad \text{where} \quad \sum_i w_i = 1$$

Here, the weights w_1,w_2,w_3 are assumed to be equal, but in general they can be tuned appropriately. The final recommendation ordering is computed by normalizing r by the importance of each co-cluster as a function of the latter's area and density.

Dataset: We used a real-world client buying pattern matrix from our institution. The matrix consists of approximately 17,000 clients and 60 product categories. Client records are further grouped according to their industry; a non-exhaustive list of industries includes: automotive, banking, travel services, education, retail, etc. Similarly, the set of products can be categorized as software, hardware, maintenance services, etc.

Graphical Interface: We built an interface to showcase the technology developed and the recommendation process. The GUI is shown in Fig.14 and consists of three panel: a) The leftmost panel displays all industry categorizations of the clients in the organization. Below it is a list of the discovered co-clusters. b) The middle panel is the co-clustered matrix of clients (rows) and products (columns). The intensity of each co-cluster box corresponds to its density (i.e., the number of black entries/bought products, over the whole co-cluster area). c) The rightmost panel offers three accordion views: the customers/products contained in the co-cluster selected; statistics on the co-cluster selected; and potential product recommendations contained in it. These are shown as red squares in each co-cluster. Note that not all white spots are shown in red color. This is because potential recommendations are further ranked by their propensity, as explained.

By selecting an industry, the user can view the co-clustered matrix and visually understand which are the most frequently bought products. These are the denser columns. Moreover, users can visually understand which products are bought together and by which customers, as revealed via the co-clustering process. Note that in the figure we purposefully have suppressed the names of customers and products, and report only generic names.

The tool offers additional visual insights in the statistics view on the rightmost panel. This functionality works as follows: when a co-cluster is selected, it identifies its customers and then hashes all their known attributes (location, industry, products bought, etc) into 'buckets'. These buckets are then resorted and displayed from most common to least common (see Figure 16). This functionality allows the users to understand additional common characteristics in the group of customers selected. For example, using this functionality marketing teams can understand what the geographical location is, in which most clients buy a particular product.

Compressibility: For the real-world data, we do not have the ground-truth of the original co-clusters, so we cannot compute the relevance of co-clusters as before. Instead, we evaluate the compressibility of the resulting matrix for each technique. Intuitively, a better co-clustered matrix will lead to higher compression. We evaluate three metrics:

- Entropy, denoted as H.
- The ratio of bytes between Run-Length-Encoding and the un-

Table 2: Comparison of compressibility of co-clustered matrix for various techniques over different metrics.

Model			Double k-Means			Inf-Theoretic[8]			MSSRCC [5]			PaCo		
Sector	Sparsity %	$K = L$	H	RLE	JPEG	H	RLE	JPEG	H	RLE	JPEG	H	RLE	JPEG
Education	7.9	5	0.8295	0.1528	0.9472	0.9215	0.1627	1	0.8356	0.1570	**0.9064**	**0.8030**	**0.1425**	0.9188
		10	0.8414	0.3063	1	0.8365	0.1435	**0.8766**	0.9034	**0.1358**	0.9044	**0.8132**	0.1387	0.8957
		15	0.9501	0.3260	0.8882	0.9787	0.1358	1	0.9171	0.1733	0.9733	**0.9077**	**0.1261**	**0.8348**
Government	12	5	0.8502	0.3068	0.9403	0.8832	**0.2480**	1	0.8703	0.2903	0.9715	**0.8036**	0.2641	**0.9184**
		10	0.8788	0.2979	0.9513	0.9908	0.2560	**0.7847**	0.9402	0.2762	1	**0.8451**	**0.2520**	0.9334
		15	0.9837	0.4444	0.9607	0.9950	0.2843	1	0.9654	0.3004	**0.8196**	**0.9474**	**0.2436**	0.8891
Industrial prod.	11.9	5	**0.8044**	0.2800	0.9719	0.9909	**0.2224**	0.9945	0.9571	0.2430	1	**0.8044**	0.2334	**0.9106**
		10	0.8258	0.3945	0.9747	0.9191	0.2571	0.9747	0.9630	0.2402	1	**0.8311**	**0.2292**	**0.9642**
		15	0.9606	0.2703	0.9619	0.9516	0.2237	1	0.9694	0.2361	0.9918	**0.9069**	**0.2045**	**0.9306**
Retail	14.2	5	0.7960	0.3558	0.9025	0.8120	0.2663	0.9788	0.8361	0.3281	1	**0.7780**	**0.2388**	**0.8734**
		10	0.8199	0.3262	1	0.9504	0.2663	**0.9108**	0.9009	0.3089	0.9678	**0.8179**	**0.2457**	0.9777
		15	0.9698	0.4302	0.9362	0.9288	0.2430	0.9205	0.9335	0.3336	1	**0.8881**	**0.2320**	**0.8995**
Services	11.4	5	0.8762	0.3342	0.8954	0.8397	**0.2378**	0.9783	0.8799	0.2636	1	**0.7198**	0.2544	**0.8333**
		10	0.9015	0.2201	0.8892	0.8841	0.2065	0.8959	0.8880	0.2507	1	**0.7423**	**0.1991**	**0.8083**

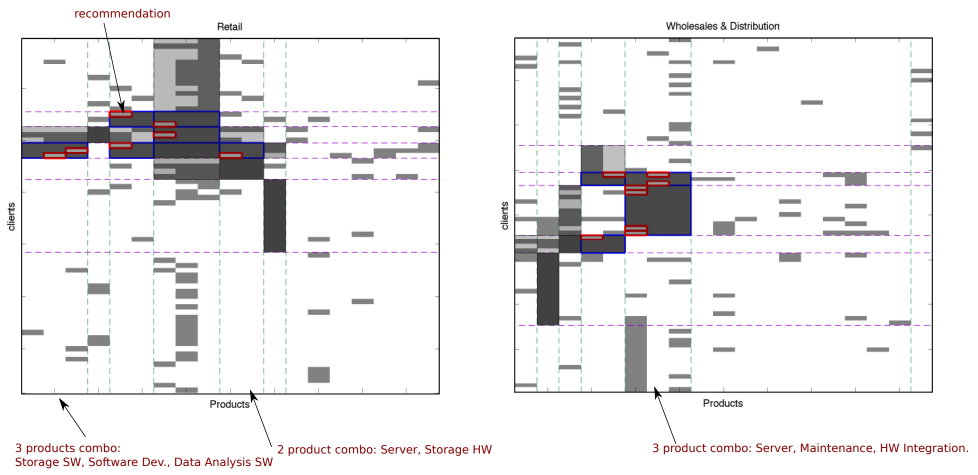

Figure 15: Examples from the coclustered matrix using PaCo for different client industries (Retail and Wholesales & Distribution). Notice the different buying pattern combos that we can discern visually.

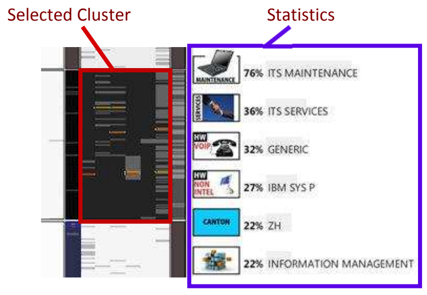

Figure 16: Summarizing the dominant statistics for a co-cluster provides a useful overview for sales and marketing teams.

compressed representation, and

- The normalized number of bytes required for compressing the image using JPEG.

The results are shown on Table 2 (lower numbers are better), and clearly suggest that PaCo provides co-clusters of the highest quality. Because it can better reorganize and co-cluster the original matrix, the resulting matrix can be compressed with higher efficiency.

Results: Figure 15 depicts some representative co-clustering examples for subsets of clients that belong to two industries: a) Wholesales & Distribution and b) Retail. We report our findings, but refrain from making explicit mentions of product names.

- For the co-clustered matrix of the Wholesales and Distribution industry, we observe that the PaCo algorithm recommends several products, identified as white spots within a dense co-cluster. The buying trend in this industry is on solutions relating to servers, maintenance of preexisting hardware and new storage hardware.

- In the matrix shown for a subset of clients in the Retail industry, we can also observe several sets of products that are bought together. Compared with the Wholesales industry, clients in this industry exhibit a different buying pattern. They buy software products that can perform statistical data analysis, as well as various server systems in the hardware area. Such a buying pattern clearly suggests that companies in the Retail industry buy combos of hardware and software that help them analyze the retail patterns of their own clients.

Comparing with Association Rules: Here, we examine the recommendation performance of co-clustering compared with that of association rules. For this experiment, we use our enterprise data and reverse 10% of 'buys' (ones) to 'non-buys' (zeros) in the original data. Then we examine how many of the flipped zeros turn up as recommendations in the co-clustering and the association rules. Note that in this experiment the notion of false positive rate is not appropriate, because some of the recommendations may indeed be potential future purchases. We measure the ratio $fc = found/changed$, i.e., how many of the true 'buys' are recovered, over the total number of buys that were changed. We also measure the ratio $fr = found/recommended$, which indicates the recovered 'buys' over the total number of recommendations offered by each technique.

We perform 1000 Monte-Carlo simulations of the bit-flipping process and report the average value of the above metrics. The results for different confidence and support levels of association rules is shown in Fig. 17. For the co-clustering using PaCo, the only parameter is the minimum density d of a co-cluster, from which recommendations are extracted. We experiment with $d = 0.7$, 0.8, and 0.9, with 0.8 being our choice for this data. We observe that PaCo can provide recommendation power equivalent to the one of association rules. For example, at confidence = 0.36 we highlight the values of the fc and fr metrics for both approaches with a red vertical line. PaCo exhibits an equivalent fc rate to that of association rules for support = 0.10, but the fr rate is significantly better than for the association rules for support = 0.10, almost equivalent with the performance at support = 0.15.

Therefore, PaCo offers equivalent or better recommendation power than association rules. Note, though, that parameter setting is more facile for our technique (only the density) compared to association rules which require the input of additional parameters (both support and confidence). More importantly, co-clustering methodologies offer superior and global view of the derived rules.

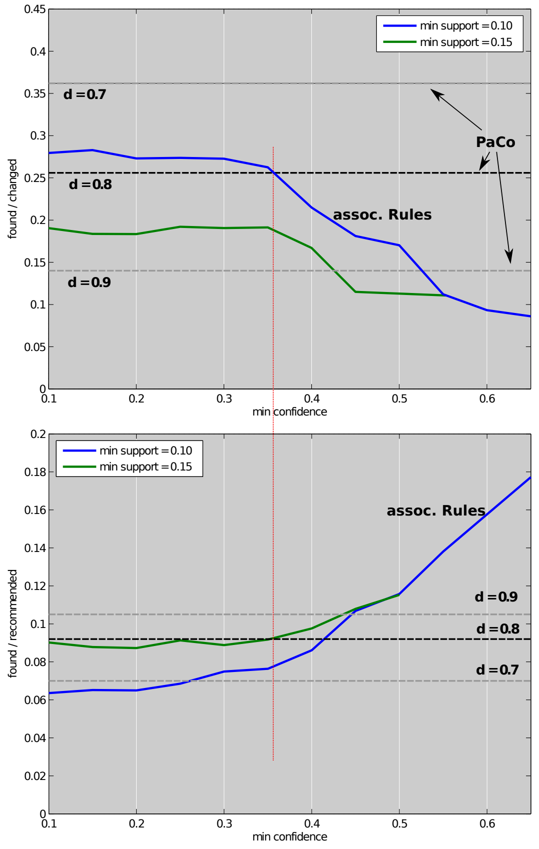

Figure 17: Comparing PaCo with association rules. Our approach provides comparable or better recommendation power, without requiring setting complex parameters.

6. CONCLUSIONS

We have introduced a scalable and noise-resilient co-clustering technique for binary matrices and bipartite graphs. Our method is inspired by both k-Means and agglomerative hierarchical clustering approaches. We have explicitly shown how our technique can be coupled with a recommendation system that merges derived co-clusters and individual customer information for ranked recommendations. In this manner, our tool can be used as an interactive springboard for examining hypotheses about product offerings. In addition, our approach can assist in the visual identification of market segments to which specific focus should be given, e.g., co-clusters with high propensity for buying emerging products, or products with high profit margin. Our framework automatically determines the number of existing co-clusters and exhibits superlative resilience to noise, as compared to state-of-the-art approaches.

As future work, we are interested in exploiting the presence of Graphical Processing Units (GPUs) for further enhancing the performance of our algorithm. There already exist several ports of k-Means on GPUs, exhibiting a speedup of 1-2 orders of magnitude, compared with their CPU counterpart [19, 13]. Such an approach represents a promising path toward making our solution support interactive visualization sessions, even for huge data instances.

7. REFERENCES

[1] A. Anagnostopoulos, A. Dasgupta, and R. Kumar. Approximation Algorithms for co-Clustering. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 201–210, 2008.

[2] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Foundations of Computer Science (FOCS)*, pages 339–348, 2005.

[3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[4] S. Bender-deMoll and D. McFarland. The art and science of dynamic network visualization. *Social Struct 7:2*, 2006.

[5] H. Cho and I. S. Dhillon. Coclustering of human cancer microarrays using minimum sum-squared residue coclustering. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 5(3):385–400, 2008.

[6] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. Minimum Sum-Squared Residue co-Clustering of Gene Expression Data. In *Proc. of SIAM Conference on Data Mining (SDM)*, 2004.

[7] I. S. Dhillon. Co-Clustering Documents and Words using Bipartite Spectral Graph Partitioning. In *Proc. of KDD*, pages 269–274, 2001.

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

[9] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. of the National Academy of Science of the United States, 95(25)*, pages 14863–14868, 1998.

[10] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *ACM Journal of Experimental Algorithmics*, 5:1, 2000.

[11] J. A. Hartigan. Direct Clustering of a Data Matrix. *J. Am. Statistical Assoc.*, 67(337):123–129, 1972.

[12] J. Kuczynski and H. Wozniakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM J. Matrix Analysis and Applications*, 13(4):1094–1122, 1992.

[13] Y. Li, K. Zhao, X. Chu, and J. Liu. Speeding up k-Means algorithm by GPUs. *J. Comput. Syst. Sci.*, 79(2):216–229, 2013.

[14] S. Madeira and A. L. Oliveira. Biclustering Algorithms for Biological Data Analysis: a survey. *Trans. on Comp. Biology and Bioinformatics*, 1(1):24–45, 2004.

[15] M. Rege, M. Dong, and F. Fotouhi. Bipartite isoperimetric graph partitioning for data co-clustering. *Data Min. Knowl. Discov.*, 16(3):276–312, 2008.

[16] H.-J. Schulz, M. John, A. Unger, and H. Schumann. Visual analysis of bipartite biological networks. *Eurographics Workshop on Visual Computing for Biomedicine*, pages 135–142, 2008.

[17] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. GraphScope: Parameter-free Mining of Large Time-evolving Graphs. In *Proc. of KDD*, pages 687–696, 2007.

[18] A. Tanay, R. Sharan, and R. Shamir. Biclustering Algorithms: a survey. *Handbook of Computational Molecular Biology*, 2004.

[19] M. Zechner and M. Granitzer. Accelerating k-means on the graphics processor via cuda. In *Proc. of Int. Conf. on Intensive Applications and Services*, pages 7–15, 2009.