High Performance Parallel/Distributed Biclustering Using Barycenter Heuristic

Arifa Nisar[*] Waseem Ahmad[†] Wei-keng Liao[‡] Alok Choudhary [§]

Abstract

Biclustering refers to simultaneous clustering of objects and their features. Use of biclustering is gaining momentum in areas such as text mining, gene expression analysis and collaborative filtering. Due to requirements for high performance in large scale data processing applications such as Collaborative filtering in E-commerce systems and large scale genome-wide gene expression analysis in microarray experiments, a high performance prallel/distributed solution for biclustering problem is highly desirable. Recently, Ahmad et al [1] showed that Bipartite Spectral Partitioning, which is a popular technique for biclustering, can be reformulated as a graph drawing problem where objective is to minimize Hall's energy of the bipartite graph representation of the input data. They showed that optimal solution to this problem is achieved when nodes are placed at the barycenter of their neighbors. In this paper, we provide a parallel algorithm for biclustering based on this formulation. We show that parallel energy minimization using barycenter heuristic is embarrassingly parallel. The challenge is to design a bicluster identification algorithm which is scalable as well as accurate. We show that our parallel implementation is not just extremely scalable, it is comparable in accuracy as well with serial implementation. We have evaluated proposed parallel biclustering algorithm with large synthetic data sets on upto 256 processors. Experimental evaluation shows large superlinear speedups, scalability and high level of accuracy.

1 Introduction

Biclustering (Subspace Simultaneous Clustering) is a very useful technique for text mining, collaborative filtering and gene expression analysis and profiling [2] [3]. Recently, it has gained increasing popularity in the analysis of gene expression data [3] . Biclustering is significantly useful and considerably harder problem than traditional clustering. Whereas a *cluster* is a set of objects with *similar* values over the *entire* set of attributes, a *bicluster* can be composed of objects with similarity over only a *subset* of attributes. Many clustering techniques such as those based on Nearest Neighbor Search are known to suffer from "Curse of Dimensionality" [4]. With large number of dimensions, similarity functions such as Euclidean distance tend to perform poorly as similarity diffuses over these dimensions. For example, in text mining, the size of keywords set describing different documents is very large and yields sparse data matrix [5]. In this case, clusters based on the entire keywords set may have no meaning for the end users.

Biclustering finds *local patterns* where a *subset* of objects might be similar to each other based on only a *subset* of attributes. The comparison between biclusters and clusters is illustrated in Figure 1. Figure 1(a) demonstrates the concept of clusters and 1(b) demonstrates the biclusters in input data matrix. Note that biclusters can cover just part of rows or columns and may overlap with each other as shown in figure 1(b). Biclustering process can generate a wide variety of object groups that capture all the significant correlation information present in a data set. Biclustering has become very popular in discovering patterns from gene microarray experiments data. Since a bicluster can identify patterns among a subset of genes based on a subset of conditions, it therefore models condition-specific patterns of co-expression. Moreover, biclustering can identify overlapping patterns thus catering to the possibility that a gene may be a member of multiple pathways.

Unprecedented growth in web related text based data repositories [6] and evolution of large biological data sets [7] has resulted into the need of high performance biclustering algorithms in order to cope with the largeness of the data sets efficiently. Given the memory size limit in single processor machines, parallel/distributed realization of the biclustering algorithms is desirable to enable scalable applications for arbitrarily large data sets.

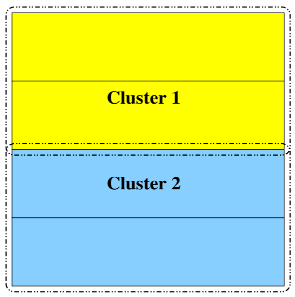
Distributed solutions for biclustering problem become essential also when the underlying data is inher-

[*]Department of Electrical Engineering and Computer Science, Northwestern University.

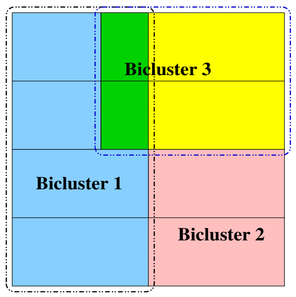
[†]Current Affiliation:A9.COM, Research was carried out when author was with ECE department at University of Illinois, Chicago.

[‡]Department of Electrical Engineering and Computer Science, Northwestern University.

[§]Department of Electrical Engineering and Computer Science, Northwestern University.



(a) Clusters of Objects.



(b) Biclusters of Objects.

Figure 1: Illustration of Clusters and Biclusters

ently distributed in nature. Consider the example of a retail store chain such as Wal-Mart who have numerous stores spread across different parts of the world. The consumer transaction data gets stored in these stores such that users are different across stores but items are generally the same. It can be viewed as row-wise distribution of the customer transaction data. Biclustering of this distributed data can be carried out in two ways. One is to transfer the data at a central location and perform subsequent sequential mining. The other way is to perform distributed biclustering of the data. The former approach has serious scalability issues as it is not feasible to transfer data from thousands of locations to a central location reliably and efficiently. Central approach is also harder to adopt because of security and privacy issues. This leaves us with the task of developing distributed solution to the biclustering problem. There has already been some work done in parallel/distributed biclustering area specially in context of document-word co-clustering [8], collaborative filtering for online recommendation systems [9] and for gene expression analysis [10].

Bipartite Spectral partitioning is a powerful technique to achieve biclustering. Normalized cut is the most popular objective function for spectral partitioning problem [11], [12]. In the absence of a polynomial time exact solution for this problem, emphasis is on finding efficient heuristics. A popular implementation [12] requires computation of the Fiedler vector for Laplacian matrix of input data. Its computation complexity is quadratic in data size and hence not suitable for large data. Recently Ahmad et al. [1] proposed a new formulation of the spectral partitioning problem as Graph Drawing(GD) problem. They showed that minimization of Hall's energy function [1] corresponds to finding the normalized cut of the bigraph. Moreover, they also showed that an optimal solution to Hall's energy minimization problem is achieved by placing each node at the barycenter of its neighbors. Based on these results, they proved that Barycenter heuristic can provide a scalable linear time solution to bipartite spectral partitioning problem. In this paper, we study the effectiveness of this approach for dealing with parallel/distributed biclustering problem in large scale datasets.

Going by the above-mentioned retail chain scenario, we assume a horizontal partitioning of the data where each location has the same feature space but objects are different. Similar assumptions were made in [8] and [9]. The parallel implementation is divided in to three stages. First stage involves parallel implementation of barycenter heuristic for crossing minimization. Identification of local biclusters is part of second stage. The third stage involves determination of global biclusters at each processor. This is accomplished by virtue of comparison of local biclusters with bicluster representatives received from other processors. A bicluster representative is merely a vector composed of means of bicluster columns over all rows of the bicluster. Each processor performs a Euclidean distance based similarity comparison with bicluster representatives received from other processors. The *similar* biclusters are subsequently merged to obtain global biclusters. Clearly during both stages of communication, the size of communication is bounded by the number of columns in the data matrix.

We show that parallel/distributed implementations for most of the existing biclustering algorithms, incur communication overhead which grows at least linearly with the size of rows. On the other hand, proposed parallel algorithm of crossing minimization based approach being independent of the number of rows, has a significant advantage for huge data sets on a large number of processors. We show that this efficient distributed implementation is comparable in accuracy to the sequential one [1]. We have evaluated the performance and accuracy of parallel biclustering algorithm on large data sets comprising of millions of rows. The evaluation results are discusses in detail in Section 6.

Organization: Rest of the paper is organized as follows. We discuss related work in section 2, section 3 talks briefly about the sequential biclustering algorithm, section 4 proposes a new model for parallel biclustering using barycenter heuristic and provides an efficient algorithm. Section 5 discusses the complexity of the proposed scheme. Experimental framework is given in Section 6, conclusions are drawn in Section 7.

2 Related Work

Recently, there have been some research efforts aimed at solving parallel biclustering problem. ParRescue [8] is one such solution which is based on Minimum Sum-Squared Residue(MSSR) clustering algorithm of Dhillon et al [13]. In our view, MSSR based biclustering approach inherently suffers from following two problems.

1. MSSR based biclustering approach is unable to determine overlapping biclusters. Overlapping biclusters are highly desired in emerging application areas such as gene expression analysis and collaborative filtering [14].

2. In this approach, row and column clusters are separately identified. It is unclear how these separately identified row and column clusters can lead to an effective biclustering solution.

3. The parallel solution incurs an $O(t(pn(k+l)))$ communication overhead where t is the total number of iterations, p is the number of processors, n is the number of columns and k and l refer to row and column clusters respectively. As we show later, this communication overhead is around $(k+l)$ times more than introduced by our implementation.

4. Finally, we believe that the MSSR based approach suffers from the initialization problem. Note that this algorithm initializes cluster and column indicator matrices randomly. This random assignment determines the quality of overall clustering solution in some cases which is a drawback. Moreover, the algorithm also requires the user to specify number of row and column clusters. Since these numbers have strong impact on final output clusters, the algorithm has undesirable dependence on these input parameters. Note that barycenter heuristic based biclustering algorithm does not suffer from these drawbacks.

Dhillon et al in [15] gave an information theoretic biclustering (Co-Clustering) algorithm. They treat the (normalized) non-negative contingency table as a joint probability distribution between two discrete random variables that take values over the rows and columns. Their subsequent work [16] provided a Bregman Divergence based loss function which was applicable to all density functions belonging to the exponential family. Bregman Divergence is defined as follows [17].

If f is a strictly convex real-valued function, the f-entropy of a discrete measure $p(x) \geq 0$ is defined by

$$H_f(p) = -\sum_x (f(p(x))$$

and the Bregman divergence $B_f(p;q)$ is given as

$$B_f(p;q) = -\sum_x f(p(x)) - f(q(x)) - \nabla f(q(x))(p(x) - q(x)) \tag{2.1}$$

When $f(x) = x \log x$, H_f is the Shannon entropy and $B_f(p;q)$ is the I-divergence, when $f(x) = -\log(x)$ we get the Burg entropy and discrete Itakura-Saito distortion $B_f(p;q) = \sum_x (\log \frac{q(x)}{f(x)} + \frac{p(x)}{q(x)} - 1)$

A parallel biclustering (co-clustering) framework was also proposed in [9] which is aimed at providing scalable solution to the collaborative filtering problem for online recommendation systems. Their contribution is based on parallelization of weighted biclustering (co-clustering) algorithm of [16]. The paper does not describe the parallel algorithm in great detail. In their theoretical analysis, they suggest that the overall computation time of the algorithm is $O(\frac{mn+mkl+nkl}{N} + Nkl)$ where N is number of processors, m and n are rows and columns respectively. Also k and l are number of column clusters and number of row clusters respectively. They have not fully discussed the communication cost of their approach. It is interesting to note that their graph on execution time with increasing number of processors (Figure 4 of [9]) shows that the execution time for `BookCrossing` dataset is around 30 seconds for a single processor case. While the execution time in case of 15 processors has decreased only by a factor of 3 to 10 seconds. This indicates that a 15 times increase in number of processors has only resulted in 3 times improvement in execution time. Moreover, the curve essentially straightens up (saturates) right after 8 processor case. Clearly the algorithm does not have a linear speed-up characteristics. We believe this is because of high communication cost incurred during accumulation of global matrix averages. On the other hand, our proposed parallel implementation of barycenter heuristic based biclustering is able to attain super-linear speedups for even hundreds of processors.

3 Biclustering using Sequential Barycenter Heuristic

3.1 Preliminaries: Throughout the paper, we denote a matrix by capital boldface letters such as **G**,**I** etc. Vectors are denoted by small boldface letters such as **p** and matrix elements are represented by small letters such as w_{ij}. Also, we denote a graph by $G(V,E)$ where V is the vertex set and E is the edge set of the graph. Moreover each edge, denoted by $\{i,j\}$, has a weight w_{ij}. The adjacency matrix of G, denoted as $\mathbf{G}$, is defined as

$$\mathbf{G} = \left\{ \begin{array}{ll} w_{ij} & \text{if there is an edge } \{i,j\} \\ 0 & \text{otherwise} \end{array} \right\}$$

DEFINITION 3.1. *Bipartite Graph: A graph $G(V,E)$ is termed as Bipartite if $V = V_0 \cup V_1$ where V_0 and V_1 are the disjoint sets of vertices (i.e. $V_0 \cap V_1 = \phi$) and each edge in E has one end point in V_0 and the other end* *point in* V_1.

We consider weighted bipartite graph $G(V_0, V_1, E, W)$ with $\mathbf{W} = (w_{ij})$ where $w_{ij} > 0$ denotes the weight of the edge $\{i, j\}$ between vertices i and j. Moreover, $w_{ij} = 0$ if there is no edge between i and j.

For above mentioned retail chain example, users are represented by vertex set V_0 and items are represented by vertices in V_1. The weight matrix $\mathbf{W}$, in this case, represents the quantity of each item bought by each user. Generally, in microarray experiment data, genes and conditions are represented by V_0 and V_1 vertex sets respectively. The edge weight w_{ij} represents the response of i'th gene to j'th condition. Similar is the case for document-word data where documents are represented by V_0 vertex set and words are represented by V_1 vertex set. In this case, the matrix $\mathbf{W}$ is the term frequency matrix for the whole corpus.

In graph theory, a cut is a partition of the vertices of a graph into two sets. Formally, for a partition of vertex set V into two subsets S and T, a *cut* can be defined as follows

$$cut(S, T) = \sum_{i \in S, j \in T} w_{ij}$$

This definition of *cut* can easily be extended to k vertex subsets,

$$cut(V_1, V_2, \ldots, V_k) = \sum_{i<j} cut(V_i, V_j)$$

It was shown in [12] that partitioning (clustering) of vertices on one layer of the bipartite graph will induce a specific clustering of vertices on the other layer which then itself induces a new clustering on the first layer. This recursive process would yield the "best" clustering of vertices on both layers when it corresponds to a partitioning of the graph such that the crossing edges between partitions have minimum weight. This essentially implies

$$\begin{aligned}\text{cut}(V_{01} \cup V_{11}, V_{02} \cup V_{12}, \ldots, V_{0k} \cup V_{1k}) = \\ min_{V_1, V_2, \ldots, V_k} cut(V_1, V_2, \ldots, V_k)\end{aligned}$$

where $V_1, V_2, \ldots, V_k$ is any partitioning of the overall Vertex set $V = V_0 \cup V_1$ into k vertex subsets. And V_{0k} and V_{1k} denote the k'th subsets of the two groups of vertices in a bipartite graph.

Min-cut may result in unbalanced partitions. The solution is to employ normalized cut instead. Minimum Normalized cut for two partitions S and T can be defined as

$$Ncut(S, T)_{min} = \frac{cut(S, T)}{vol(S)} + \frac{cut(S, T)}{vol(T)} \tag{3.2}$$

Here $vol(S)$ refers to the total weight of all edges originating from group S. Normalized cut yields more balanced partitioning.

It is well known that graph partitioning problem is NP-complete [18]. Many heuristic methods exist that can find the local minimum of the problem. Spectral graph partitioning heuristics based on finding the normalized cut of the graph are known to perform well [11] [12].

Definition 3.2. *Bipartite Drawing: A bipartite drawing of bigraph G is the embedding of its vertex sets V_0 and V_1 onto distinct points on two horizontal lines y_0, y_1 respectively while edges are drawn by straight line segments.*

Definition 3.3. *Hall's Energy Function for a Bipartite Drawing: Let $h(v_k)$ be the x-coordinate of $v_k \in V$, then the energy of bipartite drawing is defined as*

$$\mathcal{E} = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (h(v_i) - h(v_j))^2$$

Hall's energy function essentially assigns similar coordinates to vertices that are connected by edges with large weights. If the edges indicate the similarity between the vertices, minimization of Hall's energy would amount to bring similar vertices together. Intuitively, it can also be thought of as a process where graph is decomposed into clusters of vertices such that all vertices in a clusters are tightly connected to other vertices in the same cluster while they are loosely connected to vertices in other clusters. This essentially corresponds to Normalized-cut (N_{cut}). Below we give a formal proof for this intuition.

Theorem 3.4. *If $q = (q_1, q_2, \ldots, q_n)^T$ is the generalized partition vector as defined in [12], and $\boldsymbol{L}$ is the Laplacian matrix corresponding to the given graph then Hall's Energy is*

$$\mathcal{E} = \boldsymbol{q}^T \boldsymbol{L} \boldsymbol{q} = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (q_i - q_j)^2$$

Proof.

$$\begin{aligned}\mathcal{E} &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (q_i - q_j)^2 \\ &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} q_i^2 + \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} q_j^2 - \sum_{i,j=1}^{n} w_{ij} q_i q_j\end{aligned}$$

Since W is symmetric so $w_{ij} = w_{ji}$

$$\begin{aligned}\mathcal{E} &= \sum_{i,j=1}^{n} w_{ij} q_i^2 - \sum_{i,j=1}^{n} w_{ij} q_i q_j \\ &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} w_{ij} \right) q_i^2 - \sum_{i,j=1}^{n} w_{ij} q_i q_j .\end{aligned}$$

Since $w_{ii} = 0, \forall i$

$$\begin{aligned}\mathcal{E} &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} w_{ij} \right) q_i^2 - \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} w_{ij} q_i q_j \\ &= \sum_{i=1}^{n} L_{ii} q_i^2 + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} L_{ij} q_i q_j \\ &= \sum_{i,j=1}^{n} L_{ij} q_i q_j \\ &= \mathbf{q}^T \mathbf{L} \mathbf{q}\end{aligned}$$

Above theorem shows that minimization of Hall's energy corresponds to finding the normalized cut of the graph. Now we will show that assignment of x-coordinates to each node based on barycenter of its neighbors results in optimal minimum value for Hall's energy function.

Theorem 3.5. *The minimum value of Hall's energy function is achieved when*

$$h(v_i) = \frac{\sum_{j=1}^{n} w_{ij} h(v_j)}{\sum_{j=1}^{n} w_{ij}}$$

i.e. each node is assigned a new co-ordinate value which is the barycenter of its neighbors' co-ordinates.

Proof. Since Hall's energy function is a convex function, we can determine its minimum value by simply computing $\frac{\partial \mathcal{E}}{\partial h(v_i)} = 0$.

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial h(v_i)} &= \frac{\partial}{\partial h(v_i)} \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (h(v_i) - h(v_j))^2 = 0 \\ &= \sum_{i,j=1}^{n} w_{ij} (h(v_i) - h(v_j)) = 0 \\ h(v_i) &= \frac{\sum_{i,j=1}^{n} w_{ij} h(v_j)}{\sum_{j=1}^{n} w_{ij}}\end{aligned}$$

It is clear from Theorem 3.5 that a bipartite drawing which assigns each node an x-coordinate that is barycenter of its neighbors, would provide optimal solution to Hall's energy minimization problem for the given bipartite graph. Also, from Theorem 3.4, we know that Hall's energy function is equivalent to the objective function for normalized cut of the bigraph which in turn is the objective function for bipartite spectral partitioning.

Spectral clustering solutions are based on finding the Fiedler Vector (an eigenvector) of the Laplacian of the given graph. Eigenvalues and their corresponding eigenvectors are computed using techniques such as Singular Value decomposition (SVD). These techniques have typically quadratic complexity and require random access to complete data sets. This complexity limits their effectiveness for processing very large matrices.

3.2 Bigraph Crossing Minimization using Barycenter Heuristic As described above, optimal solution to Hall's energy minimization problem is attained when each vertex is placed at the barycenter (mean of ordinal values) of its neighbors. In graph drawing domain, Barycenter heuristic is a well-known technique which attempts to draw a graph on a plane such that edge crossings are minimized. It is an iterative process. During each iteration, vertex positions on one layer are fixed while the position of each vertex on other layer is computed as the mean of positions of its neighbors on the other layer. Each iteration of barycenter requires $O(|E| + |V| log|V|)$ time. For graphs with constant degree bound, the barycenter heuristic can also be implemented in linear time per iteration.

Let v_i represent the i'th node in the non-static(dynamic) layer and set N_i represent the set of neighbors of v_i. Also let r_j represent the rank of j'th member of the set N_i. Then new rank of v_i, denoted as r_i can be calculated as follows.

1. We first calculate the Weighted Mean, denoted as μ, of the ranks of neighbor nodes using Equation 3.3.
2. These weighted means represent the new ordering of the nodes. Since these means are not necessarily unique, we adjust them so that each node is assigned a unique rank based on its weighted mean.

$$\tilde{s} = \sum_{j \in N_i} w_{i,j} \times r_j \tag{3.3a}$$

$$s = \sum_{j \in N_i} w_{i,j} \tag{3.3b}$$

$$\mu_i = \frac{\tilde{s}}{s} \tag{3.3c}$$

3.3 Bicluster Identification After crossings are minimized, we perform the bicluster identification process. This process is performed on the reordered matrix representation of the Bigraph. Bicluster identification process starts from the first element of the reordered matrix and keeps on adding new columns and rows while calculating and updating a score representing the coherence of the values in the current block of the reordered matrix. This coherence is determined by virtue of two kinds of distance functions namely Bregman Divergence and Manhattan Distance.

Bregman Divergence is used to compare two rows over same set of columns. As described earlier, we can use different functions in Bregman Divergence Equation (Equation 2.1) to emulate Euclidean Distance, KL-Divergence and Itakura-Saito distortion [17]. Manhattan Distance is used to compare the values of adjacent columns in the same row. Manhattan distance for two points $P_1(x1, y1)$ and $P_2(x2, y2)$ in XY plane is given in Equation 3.4

$$D(P_1, P_2) = \mid x_1 - x_2 \mid + \mid y_1 - y_2 \mid \tag{3.4}$$

Manhattan distance is simple to compute and works well in our case as we are using it to compare distance among columns of the same row which are assumed to be identically distributed.

The bicluster identification procedure is a local search procedure. We keep pointers *StartRow* and *StartColumn* which are initialized to first row and column of the reordered matrix respectively. Coherence score for adjacent rows is calculated using Bregman Distance. Similarly the evolution pattern over adjacent columns is determined using Manhattan Distance. Our iterative procedure is row-major. We compare two rows at one time to see if they have matching columns. Starting with the *StartRow* and *Startrow* + 1, we keep a reference set of matching columns. Columns are added to this set if either of following two conditions is satisfied.

1. Bregman Distance between two rows over same set of columns is less than the given threshold δ
2. The manhattan distance between adjacent columns on both rows is the same.

The first condition makes sure that we always identify the constant value biclusters constrained by the noise. Second condition, on the other hand, guarantees that we would be able to determine the biclusters which exhibit coherent evolution over current set of columns. For each subsequent row, we find out that if it has the same coherent columns. If it has, we add it to our row set. If any row has more coherent columns than the ones in reference set, we set the *OverlapFlag* and store the current value of row iterator i in *StartRow* and that of column iterator j to *StartColumn*.

If both of the above mentioned conditions fail, we call the current submatrix a bicluster and continue with the Bicluster identification process. The next iteration of the identification process would start from *StartRow* and *Startcolumn* if *OverlapFlag* was set otherwise it could continue with the next value of row iterator till it reaches the last row. Moreover, when we reach the end of columns, we add the current row to our current Row Cluster. If we reach the end of rows, and current column and row clusters contain more than required minimum number of columns($Column_{min}$) and rows (Row_{min}) respectively, then we declare these row and column clusters to be a bicluster and add it to the Bicluster set S.

4 Parallel/Distributed Biclustering

Complete process of biclustering comprises of two main tasks; first the matrix is reordered by performing crossing minimization, followed by the identifying the biclusters in this reordered matrix. Parallel biclustering algorithm also consists of parallel crossing minimization algorithm and parallel bicluster identification. As shown in figure 2, input data matrix is horizontally partitioned amongst all the processes. After every process acquires its share of data, normalization of the data is performed to construct the joint distribution of rows and columns. In the second step, rows and columns are reordered iteratively until there is no further change. During this phase only columns' information is exchanged among the processes, rows are reordered locally. The fact that each process has to share only the column rank information is very useful in privacy preserving biclustering applications (for complete details see [14] and [19]). After reordering of local matrix, process enters in its second phase of bicluster denitrification. With the reordered matrix, each process performs a local bicluster search. After identifying the biclusters locally, bicluster representatives are shared amongst all the processes. With this new information acquired by every process, another phase of identifying local biclusters is completed. At the end, updated information is shared with all the processes and results are returned. Following sections discuss above mentioned tasks in detail.

4.1 Model for Distributed Biclustering Data is horizontally partitioned across a set $\mathcal{S} : \{s_1 \dots s_N\}$ of N Servers such that each server s_i has the same set $\mathcal{A} : \{a_1 \dots a_m\}$ of m attributes and a set $\mathcal{R}_i : \{r_{i1} \dots r_{in_i}\}$ of n_i different objects(records) such that global object

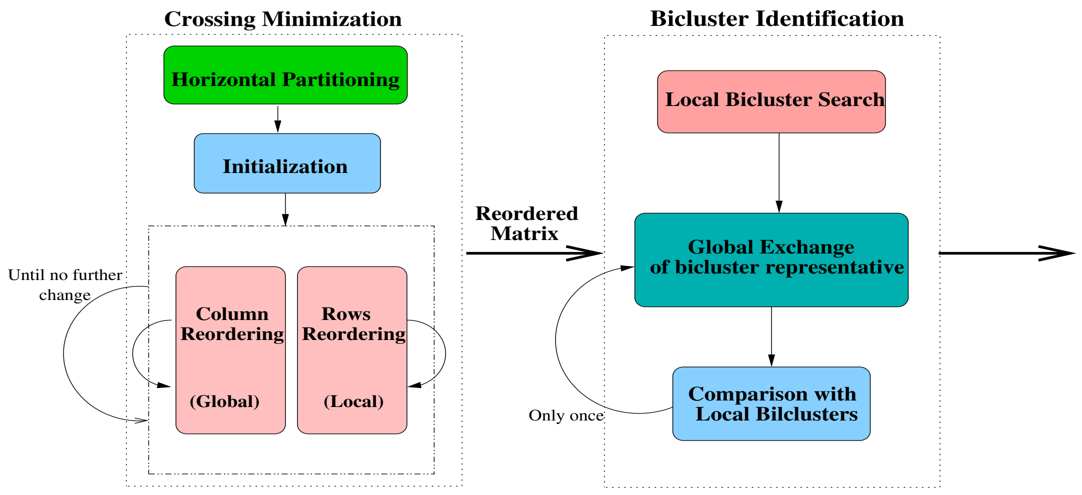


Figure 2: System Design of Parallel Biclustering Algorithm

set $\mathcal{R}$ can be represented as $\mathcal{R} = \bigcup_{\forall i \in S} \mathcal{R}_i$.

4.1.1 Parallel/Distributed Barycenter Heuristic Each server can build bigraph on its local data such that objects are represented by vertices/nodes on one layer (say $Layer0$) and the attributes/features are represented by vertices on the other layer ($Layer1$). It would then perform barycenter heuristic based crossing minimization on the Bigraph. As we can recall from previous discussion, barycenter heuristic requires iterative computation of the global ranks of object nodes and attribute nodes which are represented by two layers of the bigraph. Rank computation for each node in one layer of the bigraph requires the knowledge of ranks of its neighbors on the other layer.

Since we assume horizontal partitioning, each object node in the bigraph built over local data will have all necessary information (i.e. Ranks of its neighbor attribute nodes) to calculate its rank locally using (3.3). This implies that no inter-server communication is required for calculating the ranks of objects at each server.

On the other hand since attribute nodes are connected to object nodes which might be separated across different servers, we will have to engage in inter-server communication for exact computation of the ranks of these attribute nodes. Each server i shares its local value of μ_{ij} for j'th attribute to all other servers and then global weighted mean is calculated. This global weighted mean $\mu_G^{(j)}$ for an attribute $a_j \in A$ is simply the mean of all μ_{ij} values over n servers.

$$\mu_G^{(j)} = \frac{\sum_{1 \le i \le n} \mu_{ij}}{n}$$

The global weighted mean for an attribute node is the same over all the servers and thus results in assignment of a unique rank to each attribute node. This is an embarrassingly parallel procedure where rank of each node can be effectively computed independent of the rank of any other node in the same layer. This is specially useful for hardware based implementations of biclustering algorithm.

4.1.2 Parallel/Distributed Bicluster Identification By the end of the above mentioned distributed barycenter heuristic based crossing minimization algorithm, each server will have a reordered representation of its local data. Since ranks of object nodes are calculated locally, each server does not know the global rank of its object nodes. Global rank of object nodes is useful only if each server knows the ranks of objects on other servers too. This is because of the fact that bicluster identification procedure works on contiguously ranked object nodes. It will have to be implemented in a distributed setting by having a hash table so that each server s_i can lookup the table to determine the set of servers S_{lookup} which have object nodes adjacent to its own object node. The server s_i would then engage in distributed bicluster identification procedure in collaboration with those servers which belong to the set S_{lookup}.

It would result in tremendous increase in communication overhead. Firstly, all servers will have to engage in all-to-all broadcast of local ranks of each object node so as to assign unique global rank to each object node at each server. This is also required for building the required look up table. Given the large number of object nodes, the communication cost for this process would be prohibitively high. Secondly, during the bicluster identification process each server will have to engage in distributed bicluster identification with other servers belonging to the set S_{lookup}. This would require bicluster identification procedure to be performed in a synchronized manner at each server. Given the fact that these servers will be communicating on internet speeds and are distributed geographically, synchronization requirement will be impractical for most practical applications.

The solution to above mentioned problem is to employ a simple approach whereby object ranks are computed locally at each server and then bicluster identification process is performed locally as well. Once each server has identified its local biclusters, it broadcasts representatives of these biclusters to all other servers. A bicluster representative is a vector consisting of the mean of each attribute in the bicluster. Upon receiving the bicluster representatives from other servers, each server determines through Euclidean distance if its local biclusters can be combined with those from other servers. In case it finds strong similarity between one of its biclusters and incoming bicluster representatives. It updates the local bicluster representative for that bicluster such that the attribute means now reflect the global attribute means for the bicluster.

5 Complexity Analysis

Lets assume that $n =\mid V_0 \mid=$ total number of rows of the input data matrix,$m =\mid V_1 \mid=$ total number of columns of the input matrix and $R_c =$ average number of rows per bicluster. Also $C_c =$ average number of columns per bicluster and $C =$ total number of biclusters and $k =$ average number of iterations of barycenter heuristic.

5.1 Time complexity of Sequential Algorithm:

Now if $O(n)$ = time to compute weighted means and $O(nlogn)$ = time to perform sorting based on means and $O(n)$ = time taken in adjusting node positions. Also $O(CR_cC_c)$ = time to identify biclusters. Combining all of these components would yield sequential time complexity which is given as $T_{sequential} = O(k(n + nlog(n) + n)) + O(CR_cC_c)$. The worst case complexity for identification of a bicluster is $O(nm)$. But the identification process is carried out such that the submatrix which is part of an already identified bicluster won't have to be revisited again during identification of subsequent biclusters. In the case where a single bicluster covers the whole matrix, we will have total number of biclusters equal to one thus keeping the complexity bounded by the data size itself.

Here, without loss of generality, we can assume that $O(CR_cC_c) = O(nm)$. This implies that

$$T_{sequential} = O(k(n + nlog(n) + n)) + O(nm)$$

We have noted that k is a fairly small number for most cases. This implies that in the expression for $T_{sequetial}$, the term $O(nm)$ tends to dominate which essentially means that sequential biclustering process has complexity which is roughly the same as the problem size.

5.2 Parallel Complexity:

Lets assume that we have horizontally partitioned the data among p processors such that each processor has now n/p rows of the original matrix and all m columns. The computation complexity of scheme when data is partitioned among p processors is given by $T_p = O(k(n/p + n/plog(n/p) + n/p)) + O(nm/p)$. Going by the same arguments as in the sequential case, $O(nm/p)$ is the rough bound on parallel computation cost. This essentially means that by having p processors to work on the problem, we have reduced the overall time complexity by a factor of p which theoretically yields a linear speedup curve. Since the algorithm involves All-to-All communication primitives firstly in the crossing minimization phase (All to all reduce over Column positions), then during the cluster combining pase all-to-all gather is performed number of cluster times. All of these All-to-All communication primitives incur $O(p^2m)$ in communication cost. This indicates that the communication cost is independent of the number of rows, it only depends upon number of columns and number of processors. It grows linearly with number of columns while growing quadratically with increasing number of processors.

5.3 Impact of Memory System on Performance

As is clear from the above discussion, there are two components of proposed parallel biclustering solution. One is parallel crossing minimization and the other is bicluster identification. Both of these strongly depend on underlying memory system performance. Barycenter heuristic requires in-memory sorting of rows and columns after each iterations. We use Red-Black trees [20] to keep the nodes sorted during each iteration with respect to their current positions. Red Black trees are very useful as they provide amortized latency of $O(\log n)$ for insertion, deletion and retrieval operations. However, we observer that the cost of keeping the tree balanced increases exponentially as the data size increases beyond a certain threshold depending on the underlying memory system. This increase in tree balancing cost can be mitigated by increasing the cache size so as to reduce the I/O time spent in accessing off-chip memory and disks. This also motivates the use of parallelizing the code to make use of the caches distributed across different processors in the cluster.

The second component of parallel implementation is the bicluster identification process. This requires reading in the matrix in order determined by the above mentioned barycenter heuristic. This process is very strongly dependent on the performance of the underlying memory system based on following two reasons.

- During this procedure, matrix rows and columns are scanned based on the order determined through barycenter heuristic and not by the natural layout in the memory. This has adverse impact on the cache performance specially when the data size is very large.
- As data size increases, due to limited memory at each processor, there is high likelihood that required data might get swapped out of memory and onto the hard disk. Since I/O latency for hard disk access is orders of magnitude larger than memory access, this effect is very prominent specially in the case of sequential algorithm.

Both of the above reasons can be mitigated intrinsically by the parallel implementation. As more processors become available, partitions of large data sets can fit into their local memory more easily. Increased cache also has a strong positive impact on the performance of the parallel algorithm. In parallel processing domain, it is generally believed that super-linear speedups are attained when the parallel implementation is able to make effective use of caches of all machines in the cluster. If we consider the retail chain scenario mentioned in previous sections, this data distribution is natural. As we show in the experiment section, these reasons explain the super-linear speedups observed while comparing the performance of parallel algorithm against the sequential one.

6 Experimental Results

The proposed biclustering algorithm is implemented in MPI/C++. Our implementations for parallel biclustering algorithm is evaluated on Mercury, a machine at the National Center for Supercomputing Applications (NCSA). Mercury is an 887-node IBM Linux cluster where each node contains two Intel 1.3/1.5 GHz Itanium II processors sharing 4 GB of memory. Running a SuSE Linux operating system, the compute nodes are inter-connected by both Myrinet and Gigabit Ethernet. Implementation has been tested with MPICH version 1.2.6 on faster cpu machines (1.5GHz) with 6MB L3 cache. In all the experiments the execution time was obtained through $MPI\_Wtime()$ and is reported in seconds. The execution time doesn't take into account the time spent in reading data from files but it contains the time of writing the identified bicluster to the files.

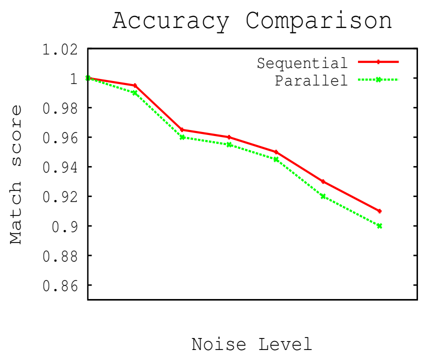


Figure 3: Accuracy Comparison between Parallel and Sequential Algorithm

6.1 Accuracy Evaluation First set of experiments are aimed at judging the accuracy of the proposed algorithm. For this purpose, we evaluate proposed parallel/distributed implementation of barycenter heuristic based biclustering algorithm against results from the sequential implementation [1]. For empirical evaluation of the quality of clusters, we use the following function which was also used in [21] and [22] .

Let $M1, M2$ be two sets of bi-clusters. The match score of $M1$ with respect to $M2$ is given by

$$S(M_1, M_2) = \frac{1}{\mid M_1 \mid} \sum_{A(I_1,J_1)\in M_1} max_{A(I_2,J_2)\in M_2} \frac{\mid I_1 \cap I_2 \mid\mid J_1 \cap J_2 \mid}{\mid I_1 \cup I_2 \mid\mid J_1 \cup J_2 \mid}$$

Let M_{opt} denote the set of implanted bi-clusters and M the set of the output bi-clusters of a biclustering algorithm. $S(M_{opt}, M)$ represents how well each of the true bi-clusters is discovered by the algorithm.

Again, we follow the approach used by Liu et al. [22] for synthetic data generation. To cater for the missing values in real life data, we add noise by replacing some elements in the matrix with random values. There are three variables b, c and γ in the generation of the bi-clusters. b and c are used to control the size of the implanted bi-cluster. γ is the noise level of the bi-cluster. The matrix with implanted constant bi-clusters is generated with four steps: (1) generate a $n \times m$ matrix A such that all elements of A are 0s, (2) generate $\sqrt{n} \times \sqrt{m}$ bi-clusters such that all elements of the biclusters are 1s, (3) implant the bi-clusters into A, (4) replace $\gamma(m \times n)$ elements of the matrix with random noise values (0 or 1) .

In the experiment, the noise level ranges from 0 to 0.25. The parameter settings used for the two methods are listed in Table 1. The resulting graph is shown

Method	Parameter Settings
Parallel	$\delta = 0.5, Iterations = 5$,Function=$KL$-Divergence,Procs=4
Sequential	$\delta = 0.5, Iterations = 5$,Function=$KL$-Divergence,Processors=1

Table 1: Parameter Settings for Parallel and sequential Biclustering Algorithm

in Figure 3. The graph illustrates the accuracy comparison between sequential barycenter heuristic based biclustering algorithm with its parallel/distributed implementation (using four processors). The graph shows that at zero noise, this scheme is almost as accurate as the sequential scheme [1]. With the increase in noise the accuracy degrades a little bit which is because of non-exact nature of bicluster comparison and merge operations of the algorithm. Note however that the accuracy of this algorithm remains considerably high even though it is a distributed implementation which uses a simple biclustering merge framework.

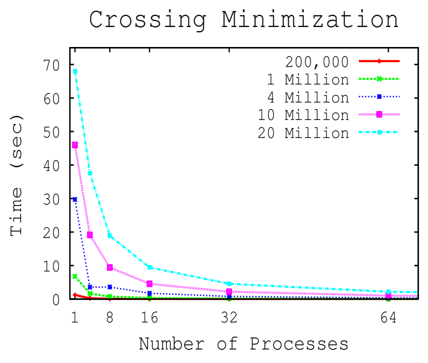
6.2 Performance Evaluation of Parallel Biclustering Algorithm Second set of experiments are aimed at analyzing the performance of parallel biclustering algorithm. For this purpose algorithm was run on synthetic data sets mentioned above. These data sets have sizes ranging from $200,000$ rows to $20,000,000$ rows, while keeping 64 columns throughout.

Figure 4 shows the performance improvement in terms of execution time for crossing minimization and bicluster identification with varying problem sizes. In figure 4(a) execution time for more than 64 processors is not reported. It is evident from these charts that bicluster identification execution time dominates crossing minimization execution time. So, total execution time shows the behavior of bicluster identification while crossing minimization becomes insignificant. Each curve represents a different data size, it is evident from the figure that execution time increases significantly with the increase in number of rows. Although for single processor, crossing minimization time increases many-fold by increasing the number of rows from $200,000$ to $20,000,000$, but figure 4(b) shows that bicluster identification case has shown extreme increase in execution time for greater than 4 Million rows. The reason for this behavior, as explained in Section 5.3 is the poor cache utilization and possible swapping of required data to secondary storage when the data size becomes too large to fit in memory. The execution time improves significantly in case of multiple processors because data partition at each processor is more likely to fit in memory and it is less likely to suffer from hard disk I/O access latencies because of swapping. Moreover, smaller data sets have better cache utilization as there is less cache pollution. Since there is orders of magnitude difference between I/O latencies from cache and those from hard disk, the end result is exponential reduction in executions times for multiple processor cases. We are not presenting the execution time more than 5 hours in this paper to avoid imbalanced scaling in charts. For bigger data sizes, execution time with only large number of processors is reported in figure 4, because execution time is considerably high for smaller number of processes.

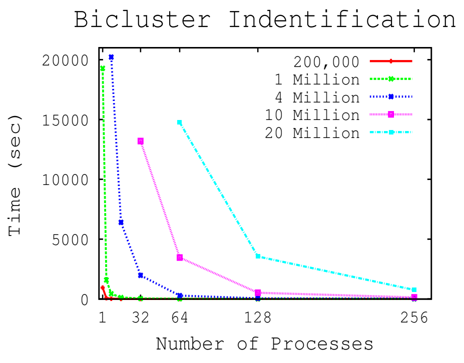
Figure 5 shows the charts for speedup, which is defined as $S = \frac{Time(p=1)}{time(p=P)}$, for different problem sizes from 1 to 256 processors. First graph shows speedup curves for parallel crossing minimization, all other graphs show overall speedup achieved for different problem sizes. For $200,000$ rows speedup decreases as the number of processors is increased beyond 64. This is because of the fact that inter-processor communication cost which is $O(p^2)$ starts undermining the performance gains made through data decomposition and resulting parallel execution of computation tasks. Similar phenomenon could be seen for other problem sizes as well if the number of processors are increased further. 1 Million rows case also seems to saturate after 256 processors. Smaller the problem size, sooner saturation will occur. Note that for the cases of 4 Million, 10 Million and 20 Million rows, speedup has been calculated with respect to 8, 32 and 64 processors respectively. It is clear from the graphs that the proposed approach is very scalable and maintains excellent speedup characteristics throughout. As discussed earlier, superlinear speedups are result of better cache utilization with smaller data sizes and avoidance of high latency secondary storage I/O.

7 Conclusions

Parallel/distributed implementation of biclustering algorithm is highly desired in many application domains. These domains include gene expression analysis in large scale micro-array experiments, text mining on large web based text repositories and collaborative filtering in E-commerce systems. In order to meet this challenge, a parallel/distributed algorithm for barycenter heuristic based biclustering algorithm is proposed. We provide thorough analysis of the advantages of this scheme over existing solutions. Proposed approach is evaluated on synthetic data sets of large sizes on a cluster with hundreds of machines. Both accuracy and performance of the algorithm are tested and verified for these data sets. The experiments reveal that the algorithm not only shows super-linear and scalable speedup charac-



(a)



(b)

Figure 4: Execution Time for Crossing Minimization and Bicluster Identification for Different Problem Sizes.

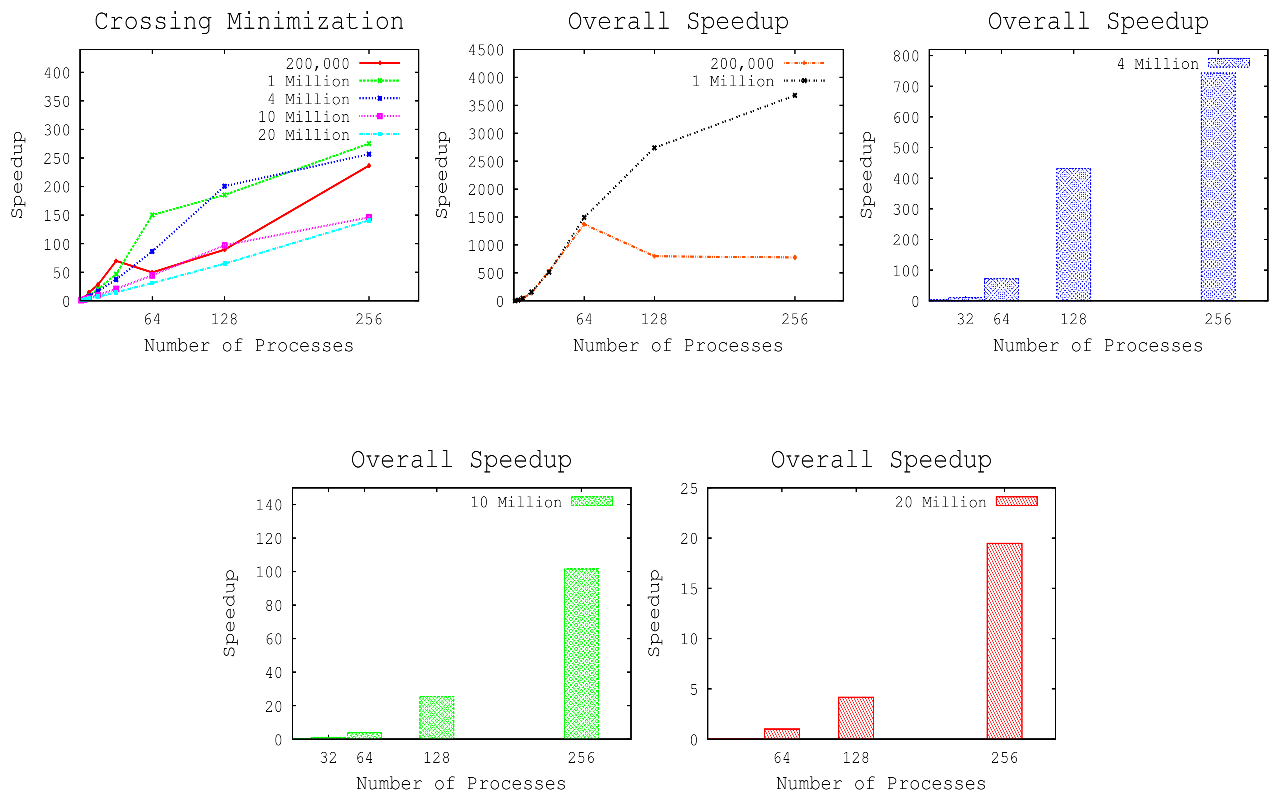


Figure 5: Crossing Minimization's and Overall Speedup for Different Problem Sizes.

teristics but also maintains good accuracy throughout.

8 Acknowledgements

This work was supported in part by DOE SCIDAC-2: Scientific Data Management Center for Enabling Technologies (CET) grant DE-FC02-07ER25808, DOE SCiDAC award number DE-FC02-01ER25485, NSF HECURA CCF-0621443, NSF SDCI OCI-0724599, NSF CNS-0551639 and IIS-0536994. This research was supported in part by the National Science Foundation through TeraGrid resources provided by NCSA, grant numbers TG-CCR060017T, TG-CCR080019T, and TG-ASC080050N. We gratefully acknowledge Prof. Ashfaq khokar of University of Illinois, Chicago and anonymous reviewers for their insightful suggestions and helpful comments.

References

[1] W. Ahmad and A. Khokhar, "chawk: A highly efficient biclustering algorithm using bigraph crossing minimization," in *Second International Workshop on Data Mining and Bioinformatics, VDMB 2007, Vienna, Austria (In conjunction with VLDB2007)*, 2007.

[2] Y. Cheng and G. Church, "Biclustering of expression data," in *Proceedings of Intelligent Systems for Molecular Biology*, 2000.

[3] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1, 2004.

[4] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[5] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine Learning*, vol. V42, no. 1, pp. 143–175, January 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1007612920971

[6] A. Kittur, E. Chi, B. A. Pendleton, B. Suh, and T. Mytkowicz, "Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie," *World Wide Web*, vol. 1, no. 2, 2006.

[7] K. hoi Cheung, K. White, J. Hager, M. Gerstein, and M. S. P. P. Miller, "Ymd: a microarray database for large-scale gene expression analysis," in *Proc AMIA Symp*, 2002, pp. 140–144.

[8] J. Zhou and A. A. Khokhar, "Parrescue: Scalable parallel algorithm and implementation for biclustering over large distributed datasets." in *ICDCS*. IEEE Computer Society, 2006, p. 21.

[9] T. George and S. Merugu, "A scalable collaborative filtering framework based on co-clustering." in *ICDM*. IEEE Computer Society, 2005, pp. 625–628.

[10] L. Wei, C. Ling, Q. Hongyu, and Q. Ling, "A parallel biclustering algorithm for gene expressing data." IEEE Computer Society Press, 2008.

[11] M. Gu, H. Zha, C. Ding, X. He, and H. Simon, "Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering," 2001. [Online]. Available: citeseer.ist.psu.edu/article/gu01spectral.html

[12] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Knowledge Discovery and Data Mining*, 2001, pp. 269–274. [Online]. Available: citeseer.ist.psu.edu/dhillon01coclustering.html

[13] Y. G. H. Cho, I. S. Dhillon and S. Sra, "Minimum sum-squared residue co-clustering of gene expression data," in *Proceedings of the fourth SIAM International Conference on Data Mining*, 2004, pp. 114–125.

[14] W. Ahmad and A. Khokhar, "An architecture for privacy preserving collaborative filtering on web portals," in *The Third International Symposium on Information Assurance and Security (IAS)*, 2007.

[15] S. M. I. S. Dhillon and D. S. Modha., "Information-theoretic co-clustering," in *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Miing (KDD)*, 2003, pp. 89–98.

[16] I. S. D. A. Banerjee, S. Merugu and J. Ghosh, "Clustering with bregman divergences," *Journal of Machine Learning Research*, vol. 6, pp. 1705–1749, 2005.

[17] J. Lafferty, S. Pietra, and V. Pietra, "Statistical learning algorithms based on bregman distances," in *Proceedings of the Canadian Workshop on Information Theory, 1997.*, 1997.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[19] W. Ahmad and A. A. Khokhar, "Phoenix: Privacy preserving biclustering on horizontally partitioned data," in *Privacy, Security, and Trust in KDD, First ACM SIGKDD International Workshop, PinKDD 2007, San Jose, CA, USA, August 12, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 4890. Springer, 2008, pp. 14–32.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[21] A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Buhlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler, "A systematic comparison and evaluation of biclustering methods for gene expression data," *Bioinformatics*, 2006.

[22] L. W. X. Liu, "Computing the maximum similarity bi-clusters of gene expression data," *Bioinformatics*, vol. 23, no. 1, pp. 50–56, 2007.