Exemplar-based Robust Coherent Biclustering

Kewei Tu * Xixiu Ouyang † Dingyi Han ‡ Yong Yu § Vasant Honavar ¶

Abstract

The biclustering, co-clustering, or subspace clustering problem involves simultaneously grouping the rows and columns of a data matrix to uncover biclusters or sub-matrices of the data matrix that optimize a desired objective function. In coherent biclustering, the objective function contains a coherence measure of the biclusters. We introduce a novel formulation of the coherent biclustering problem and use it to derive two algorithms. The first algorithm is based on loopy message passing; and the second relies on a greedy strategy yielding an algorithm that is significantly faster than the first. A distinguishing feature of these algorithms is that they identify an exemplar or a prototypical member of each bicluster. We note the interference from background elements in biclustering, and offer a means to circumvent such interference using additional regularization. Our experiments with synthetic as well as real-world datasets show that our algorithms are competitive with the current state-of-the-art algorithms for finding coherent biclusters.

1 Introduction

The biclustering, co-clustering, two-way clustering, or subspace clustering problem involves simultaneously grouping the rows and columns of a data matrix to uncover biclusters or sub-matrices of the data matrix that optimize a desired objective function. Sometimes these names refer to different variants of the biclustering problem: for example, the term "co-clustering" was used in [9] to describe the problem of finding biclusters that form a checkerboard pattern in the data matrix. Applications of biclustering include: finding groups of genes that display similar expression patterns under subsets of time points or conditions [6, 13, 25, 24, 21, 7, 8]; finding groups of users who share interest in certain subsets of movies [25, 24]; finding clusters of documents that share subsets of words [9]; and finding correlations between groups of words and phrases in a natural language corpus to induce grammar rules [1, 23].

The first algorithm for simultaneous clustering of the rows and columns of a data matrix was introduced by Hartigan [14] . In recent years, there has been a growing interest in biclustering algorithms, especially those algorithms capable of finding coherent biclusters (see [17] for a survey). One of the first algorithms for coherent biclustering was described by Cheng and Church [6], who define bicluster coherence as the mean squared residue of the data matrix elements assigned to a bicluster, and identify biclusters one at a time using greedy search. The algorithm introduced by Yang et al. [25] starts with a set of random biclusters and iteratively expands or shrinks them in a greedy fashion. The coherent biclustering algorithm of Cho et al. [7] attempts to find coherent biclusters positioned in a checkerboard pattern. Lazzeroni and Owen [16] introduced a probabilistic model (the plaid model) to describe arbitrarily positioned and possibly overlapping coherent biclusters. Recent work has explored nonparametric Bayesian models for biclustering [19, 15]. Deodhar et al. [8] have recently introduced ROCC, a scalable and noise-tolerant biclustering algorithm that can discover an a priori unknown number of arbitrarily positioned, possibly overlapping, coherent biclusters.

Against this background, this paper introduces a novel formulation of coherent biclustering and uses it to derive two algorithms for solving the problem. The first algorithm uses a message passing technique[11] to optimize the coherence measure; the second employs a greedy strategy resulting in an algorithm that is significantly faster and hence much more scalable than the first. The proposed algorithms *simultaneously* find all the coherent biclusters from a data matrix. They are robust in the presence of *noise* as well as *missing* elements in the data matrix. Our first algorithm finds non-overlapping (disjoint) biclusters, whereas our second algorithm can cope with overlapping biclusters as well. Unlike many existing algorithms that assume a checkerboard pattern of the biclusters [7, 9], our algorithms can find biclusters that are arbitrarily positioned, a desirable feature in applications where an object (a row

*tukw@iastate.edu, Department of Computer Science, Iowa State University, Ames, IA 50011, USA.

†xxouyang@apex.sjtu.edu.cn, Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China.

‡handy@apex.sjtu.edu.cn, Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China.

§yyu@apex.sjtu.edu.cn, Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China.

¶honavar@cs.iastate.edu, Department of Computer Science, Iowa State University, Ames, IA 50011, USA.

or a column of the data matrix) may belong to multiple groups. Our algorithms can be used to find not only coherent biclusters, but also other types of biclusters such as constant-value, constant-row, or constant-column biclusters. They can utilize different optimization criteria based on different models of noise in the data. Furthermore, they do not require that the number of biclusters be pre-specified. To the best of our knowledge, with the exception of the ROCC algorithm [8], few biclustering algorithms simultaneously offer all of these desirable features.

In addition to the features enumerated above, the proposed algorithms offer two distinct advantages over existing approaches. The first advantage is that they identify an exemplar element (i.e., a representative row-column pair) of each bicluster. The exemplars facilitate the process of optimizing the objective function of biclustering; assist in the interpretation of the resulting biclusters; and contribute to the increased robustness of the algorithms in the presence of outliers (as compared with existing approaches). The second advantage of our algorithms is their robustness in the presence of interference from background elements. Such interference occurs if a row or column of background elements can fit into a coherent bicluster according to the coherence measure. For example, in the case of multiplicative coherence measure, adding a row or column of zeros can improve the coherence of a bicluster; but in many applications zero is the default value for the elements of the data matrix, and hence it makes little sense to include a row or column of zeros which carry little useful information into a bicluster. This problem is especially pronounced in applications in which the data matrix is typically sparse (containing a large number of background elements), as in the case of a word bigram matrix in natural language processing applications. The problem of background interference cannot be solved by simply excluding background-value elements from biclusters because a bicluster can sometimes contain such elements. Moreover, because biclustering typically involves iterative optimization, background interference needs to be dealt with *during* the optimization process. Dealing with background interference is further complicated by noise in the data matrix which makes it hard to reliably distinguish background elements (e.g., values close to zero) from the rest of the elements of the data matrix. We solve the background interference problem by introducing an additional regularization term into the objective function that drives biclustering.

Section 2 introduces our mathematical formulation of the biclustering problem. Section 3 and 4 (respectively) describe our message passing algorithm and our greedy algorithm for biclustering. Section 5 presents results of experiments comparing the biclustering algorithms proposed in this paper with a few state-of-the-art biclustering algorithms available in the literature. Section 6 concludes with a summary of the paper.

2 Exemplar-based Robust Coherent Biclustering

2.1 Bicluster Coherence. Given a data matrix A, a bicluster is a submatrix of A that satisfies some desired properties. A coherent bicluster is a bicluster in which the value of each element is consistently a function of the row and the column of the element. That is, ideally the element at row i and column j of the bicluster has value

$$a_{ij} = F(\alpha_i, \beta_j, \mu)$$

where α_i is a constant specific to row i, β_j is a constant specific to column j, μ is a constant specific to the bicluster, and F is some function. Two types of coherence that have been studied in the literature include:

$$\begin{aligned} \text{additive coherence:} &\quad F(\alpha_i, \beta_j, \mu) = \mu + \alpha_i + \beta_j \\ \text{multiplicative coherence:} &\quad F(\alpha_i, \beta_j, \mu) = \mu \times \alpha_i \times \beta_j \end{aligned}$$

Other simpler types of biclusters, e.g., constant-value biclusters, constant-row biclusters, and constant-column biclusters, are all special cases of coherent biclusters (see [17] for details). Note that in general, elements of the data matrix A and hence elements of biclusters need not be scalars. For example, in [23] each element of A is a vector and multiplicative coherence is defined with μ being a bicluster-specific constant vector.

To measure the coherence of a bicluster, we need to compare the actual value of each element a_{ij} with its ideal value a^*_{ij}. Since the values of α_i, β_j and μ are a priori unknown, we have to estimate a^*_{ij} from the data. In most of the previous work, a^*_{ij} is estimated using some statistics of the bicluster, e.g., the mean of row i, the mean of column j and the mean of the bicluster. In this paper, we adopt a different approach to estimating a^*_{ij}. Within each bicluster, we specify an exemplar element, say a_{kl} located at row k and column l. Then we have:

$$\begin{aligned} \text{for additive coherence:} &\quad a^*_{ij} = a_{il} + a_{kj} - a_{kl} \\ \text{for multiplicative coherence:} &\quad a^*_{ij} = a_{il} \times a_{kj} \div a_{kl} \end{aligned}$$

The value of a^*_{ij} for other types of coherence or special cases of coherence can be formulated in a similar fashion. It is easy to see that, for a perfectly coherent bicluster, this method yields exactly the same result as the traditional method using mean values. However, when the bicluster is not perfectly coherent, these two methods behave differently. In settings where the bicluster

contains relatively few elements and has significant levels of noise, the mean-based estimates of a^*_{ij} are likely to be more robust than the exemplar-based estimates; in settings where there are outliers in the bicluster, the exemplar-based estimates of a^*_{ij} are likely to be more robust than their mean-based counterparts. The coherence measure defined using exemplar-based estimates of a^*_{ij} induces relatively weak coupling between the elements of a bicluster, in comparison with the coherence measure defined using mean-based estimates. As we shall see, this simplifies the task of optimizing the coherence measure. Furthermore, the chosen exemplar of a bicluster, i.e., a representative row-column pair, can be useful in interpreting the bicluster.

Once the ideal values are estimated for all the elements in the bicluster, we use the sum of the distances between the actual and ideal values as the coherence measure of the bicluster. Given a data matrix A, a biclustering algorithm outputs a set B of biclusters contained in A, whose overall coherence is given by:

$$K(B) = \sum_{b \in B} \sum_{\substack{i \in b.\text{rows} \\ j \in b.\text{columns}}} w_{ij}\, d(a_{ij}, a^*_{ij})$$

where a_{ij} and a^*_{ij} are (respectively) the actual and ideal values of the element at row i and column j that is assigned to a bicluster b; d is a suitable distance measure defined in accordance with the model of noise in the data; w_{ij} is a non-negative weight that is used to cope with missing values or imprecision in the data: it is set to zero if the value of a_{ij} is unknown; and it is set to a small positive value if the value of a_{ij} has a high degree of imprecision associated with it. Here if the distance measure d is defined to be the squared difference between a_{ij} and a^*_{ij}, and w_{ij} is set to 1 for any i, j, then $K(B)$ reduces to the *total squared residue* defined in [7]. However, a variety of other distance measures can be used, e.g., those described in [3].

Now suppose we require that each element in the data matrix A must belong to exactly one bicluster in B (so B corresponds to a partition of the elements in the data matrix A). Recall that each bicluster in B has an exemplar, so we can associate each element a_{ij} of the data matrix A with the exemplar of the bicluster that a_{ij} belongs to. Let c_{ij} be the index of the exemplar of a_{ij}. Suppose the data matrix A is of the size $m \times n$, then $\mathbf{c} = (c_{11}, c_{12}, \ldots, c_{mn})$ defines a biclustering configuration, i.e., an assignment of the elements of the data matrix A to a set of disjoint biclusters (or equivalently, bicluster exemplars). Now we can rewrite the coherence measure $K(B)$ to an energy function of the corresponding biclustering configuration $\mathbf{c}$:

$$E(\mathbf{c}) = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij}\, d(a_{ij}, a^*_{ij})$$

where a^*_{ij}, the ideal value of a_{ij}, is computed as described before, based on the exemplar of a_{ij} as specified by the index c_{ij}. In many biclustering applications, however, it is not reasonable to insist that every element of the data matrix A be assigned to a bicluster. For instance, it is desirable to exclude the background (e.g., zero-valued) elements of A from the biclusters. We will deal with this issue in the next subsection.

2.2 Regularization. Optimizing the overall coherence of biclustering alone is likely to result in overfitting, so we introduce two regularization terms in this subsection.

First, we add a penalty term that grows in proportion to the number of biclusters. This helps avoid getting too many small or even single-element biclusters. Since each bicluster has exactly one exemplar, penalizing the number of biclusters is equivalent to penalizing each exemplar by a positive penalty term p_1 (with one exception, explained in the next paragraph). Based on our experience, p_1 should be assigned a value close to the average distance between the actual and ideal element values in the data matrix. Note that if we have an a priori preference of some elements being or not being exemplars, we can also customize the value of p_1 for such elements when they serve as exemplars.

Since we require each element to belong to some bicluster, many biclusters may consist of only background elements (typically zero or close to zero). Such biclusters carry little useful information, so it is reasonable to ignore them in calculating the penalty based on the number of biclusters. To avoid penalizing such biclusters while keeping the energy function simple, first we disallow any background element from being selected as an exemplar for any element other than itself (to achieve this, in computing the coherence measure we force the distance between the actual and ideal values of an element to be infinity if the exemplar of this element is a background element other than itself). Then, for a background element exemplar (which can only be the exemplar of itself), we assign a penalty ϵ that is typically set to zero but can also be set to a negative value (thus becoming a reward, not a penalty) if we know a priori that the biclusters in the data matrix are unlikely to contain any background element.

Second, we add a small penalty based on the size of each bicluster, in order to avoid a non-background bicluster from incorporating any row or column consisting of only background elements. Specifically, for each

bicluster we set this penalty as the sum of the number of rows and the number of columns of the bicluster multiplied by a small positive constant $p_2 \ll p_1$. This helps avoid including any row or column of background elements into a bicluster, because doing so will increase the bicluster size and thus incur the penalty, whereas there would be no penalty if the row or column of background elements get assigned to biclusters consisting of only themselves (as explained above).

Note that the penalty based on bicluster size would not exclude any row or column of non-background elements that do belong to the bicluster, because p_1 that penalizes the number of biclusters is much larger than p_2 that penalizes the size of biclusters. In the case of a data matrix containing a high level of noise, it can be difficult to reliably distinguish between background and non-background values of a data matrix. In such settings, we can modify the penalty p_2 to be proportional to the probability that a row or column consists of only background elements.

The second penalty term that penalizes bicluster sizes is especially useful in settings where the data matrix is sparse (i.e., contains a large number of background elements) and background elements can fit into a coherent bicluster according to the coherence measure. For example, in the absence of this penalty, a row or column of zeros can be added into a bicluster of multiplicative coherence while do not hurt, or even improve, the bicluster coherence; however, in settings where background elements of the data matrix take the value zero, such addition of rows or columns of zeros into a bicluster can result in misleading and potentially erroneous conclusions.

If an element in the data matrix selects itself as its exemplar, then its ideal value a^* is exactly the same as its actual value a and the distance $d(a, a^*)$ is 0. Therefore we can rewrite the energy function with the two regularization terms included in the following form:

$$E(\mathbf{c}) = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathrm{CP}_{ij}(c_{ij})$$

where CP_{ij} is a *Coherence&Prior* matrix that combines the coherence measure and the two regularization terms, defined as follows. Suppose the value of c_{ij} is (k, l), then if a_{kl} is a background-value element,

$$\mathrm{CP}_{ij}((k,l)) = \begin{cases} +\infty & \text{if } (k,l) \neq (i,j) \\ \epsilon & \text{if } (k,l) = (i,j) \end{cases}$$

and if a_{kl} is not a background-value element,

$$\mathrm{CP}_{ij}((k,l)) = \begin{cases} w_{ij}\, d(a_{ij}, a^*_{ij}) & \text{if } k \neq i \text{ and } l \neq j \\ p_2 \quad \text{if } k = i \text{ or } l = j \text{ but } (k,l) \neq (i,j) \\ p_1 + 2p_2 & \text{if } (k,l) = (i,j) \end{cases}$$

As mentioned earlier, we can replace p_1 with a value customized for each element based on some a priori preference as to whether the element in question should be an exemplar; and we can replace p_2 with a value based on the probability of a row or column containing only background elements.

2.3 Constraints. We now introduce two constraints on the minimization of the energy function used to drive biclustering. The first constraint is that, for any two rows i, k and two columns j, l, if element a_{ij} and element a_{kl} are associated with the same exemplar, or if element a_{il} and element a_{kj} are associated with the same exemplar, then all of the four elements must be assigned to the same exemplar. This constraint ensures that the set of elements that choose the same exemplar must constitute a valid bicluster, i.e., a submatrix of the data matrix A. The second constraint is that if the element a_{ij} is an exemplar for some element, then a_{ij} must become its own exemplar. This ensures that the exemplar of a bicluster necessarily belongs to the bicluster.

We can combine the energy function $E(\mathbf{c})$ and the two constraints into a new objective function $S(\mathbf{c})$, such that if the configuration $\mathbf{c} = (c_{11}, c_{12}, \ldots, c_{mn})$ violates the constraints, then the function value is $-\infty$; otherwise it is the negative of the energy:

$$\begin{aligned} S(\mathbf{c}) \;=\; & -E(\mathbf{c}) + \sum_{1\le i<k\le m}\sum_{1\le j<l\le n} f_{ijkl}(\mathbf{c}) \\ & + \sum_{1\le i\le k\le m}\sum_{1\le j\le n}\sum_{\substack{1\le l\le n \\ \text{s.t. if } i=k \text{ then } j<l}} g_{ijkl}(\mathbf{c}) \end{aligned}$$

f is the constraint function for the first constraint.

$$f_{ijkl}(\mathbf{c}) = \begin{cases} -\infty & \text{if } (c_{ij} = c_{kl} \neq c_{kj} \text{ or } \neq c_{il}) \\ & \quad \text{or } (c_{il} = c_{kj} \neq c_{ij} \text{ or } \neq c_{kl}) \\ 0 & \text{otherwise} \end{cases}$$

g is the constraint function for the second constraint.

$$g_{ijkl}(\mathbf{c}) = \begin{cases} -\infty & \text{if } (c_{ij} = (k,l) \text{ but } c_{kl} \neq (k,l)) \\ & \quad \text{or } (c_{kl} = (i,j) \text{ but } c_{ij} \neq (i,j)) \\ 0 & \text{otherwise} \end{cases}$$

This objective function can be represented by a factor graph[5], as shown in Fig.1.

3 An Exemplar-Based Biclustering Algorithm via Message Passing

We now proceed to introduce our exemplar-based biclustering algorithm using message passing (EBMP).

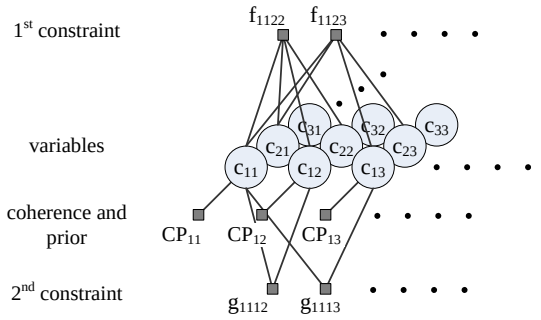


Figure 1: The factor graph representing the factorization of the objective function. Each circle represents a variable. Each box is a factor, connected to the variables it depends on.

3.1 The Basic Algorithm. Because exact optimization on the factor graph shown in Fig.1 is intractable, we resort to approximate inference. We can apply the max-sum algorithm[5] here, which iteratively propagates messages between factors and variables and usually converges to an approximate optimum. However, the time complexity of direct application of the max-sum algorithm on this factor graph is exponential in the size of the data matrix. Hence, we derive below an equivalent algorithm that is polynomial in the size of the data matrix.

In the max-sum algorithm, the update rule for a message from a factor h to a variable c_{ij} is given by:

$$\alpha_{h\to c_{ij}}(c) = \max_{c_x,\ldots,c_y}\left[h(c,c_x,\ldots,c_y) + \sum_{z=x,\ldots,y}\rho_{c_z\to h}(c_z)\right]$$

where $c_x,\ldots,c_y$ are the variables connected to h other than c_{ij}, and $\rho_{c_z\to h}(c_z)$ denotes the message from the variable c_z to the factor h. We have three types of factors, which we consider in turn.

First, if h is CP_{ij}, since it is connected to only one variable c_{ij}, the message is always equal to the factor $\mathrm{CP}_{ij}(c)$.

Second, if h is f_{ijkl}, note that $f_{ijkl}(\mathbf{c})$ is either $-\infty$ or 0, so because of the max operation we need to consider only the configurations that make this quantity 0 (i.e., configurations that satisfy the first constraint); then we can exchange the order of max and sum, resulting in the following simplified formula. Suppose f is connected to the variables $c_{ij}, c_{il}, c_{kj}, c_{kl}$, where i,k are row indices and j,l are column indices.

(3.1)
$$\alpha_{f\to c_{ij}}(c) = \max\left[\begin{array}{c}\max_{c'\neq c}\rho_{c_{kl}\to f}(c') + \max_{c'\neq c''}(\rho_{c_{il}\to f}(c') + \rho_{c_{kj}\to f}(c'')),\\ \rho_{c_{kl}\to f}(c) + \rho_{c_{il}\to f}(c) + \rho_{c_{kj}\to f}(c)\end{array}\right]$$

There are two expressions in the max operation. The first corresponds to the configuration where neither of the two diagonal pairs of variables are equivalent, and the second corresponds to the configuration where all four variables are equivalent. It is easy to show that, there are only two cases in which the second expression may have a larger value than the first. First, when $c = \arg\max(\rho_{c_{kl}})$; second, when $c = \arg\max(\rho_{c_{il}}) = \arg\max(\rho_{c_{kj}})$. With the exception of the preceding two cases, the value of $\alpha_{f\to c_{ij}}(c)$ is a constant regardless of the value of c, which we denote by $\bar{\alpha}_{f\to c_{ij}}$.

$$\bar{\alpha}_{f\to c_{ij}} = \max_{c'}\rho_{c_{kl}\to f}(c') + \max_{c'\neq c''}\left(\rho_{c_{il}\to f}(c') + \rho_{c_{kj}\to f}(c'')\right)$$

Third, if h is g_{ijkl}, we can simplify the update formula in a similar way. Suppose factor g is connected to the variables c_{ij}, c_{kl}.

(3.2)
$$\alpha_{g\to c_{ij}}(c) = \begin{cases}\max_{c'}\rho_{c_{kl}\to g}(c') & \text{if } c=(i,j)\\ \rho_{c_{kl}\to g}(c) & \text{if } c=(k,l)\\ \max_{c'\neq(i,j)}\rho_{c_{kl}\to g}(c') & \text{otherwise}\end{cases}$$

Again, the message value is a constant except in two special cases, i.e., when $c=(i,j)$ and when $c=(k,l)$. Denote this constant by $\bar{\alpha}_{g\to c_{ij}}$.

$$\bar{\alpha}_{g\to c_{ij}} = \max_{c'\neq(i,j)}\rho_{c_{kl}\to g}(c')$$

In summary, each message from a constraint factor to a variable can be summarized by a constant and two special cases. We further apply a normalization step that subtracts the constant $\bar{\alpha}_{h\to c_{ij}}$ from the values of each message $\alpha_{h\to c_{ij}}(c)$, so that the normalized $\alpha_{h\to c_{ij}}(c)$ is zero for any value of c except the two special cases. The purpose of this normalization is to avoid overflow, because all the message values are initially less than or equal to 0 and are iteratively updated to the sum of one or more other messages, leading to exponential increase in magnitude in the absence of normalization. It is easy to show that this normalization does not alter the output of the algorithm.

The update rule for a message from variable c_{ij} to factor h is simply the sum of all the incoming messages of c_{ij} minus the message from h.

(3.3)
$$\rho_{c_{ij}\to h}(c) = \sum_{h'\in\mathrm{ne}(c_{ij})\setminus h}\alpha_{h'\to c_{ij}}(c)$$

where $\mathrm{ne}(c_{ij})$ is the set of factors connected to c_{ij}.

Now that we have specified all the message update rules, we need a message passing schedule to determine the order in which the messages are updated. The naive round-robin schedule usually fails to converge in reasonable time or yields poor results. Hence, we use *residual*

belief propagation (RBP)[10], an informed schedule that orders the messages by how much they change as a result of update and always updates the message with the largest change first. To avoid oscillation, when updating a message we damp it so that the new message value is a weighted average of its previous and current values.

We initialize all the messages to zero, and then iteratively update them according to the RBP schedule, until the biclustering configuration $\mathbf{c}$ stays unchanged for a number of iterations, or when a pre-specified maximal number of iterations is reached. The configuration $\mathbf{c}$ can be read out by setting each c_{ij} to

$$\arg\max_{c} \sum_{h \in \mathrm{ne}(c_{ij})} \alpha_{h \to c_{ij}}(c)$$

where $\mathrm{ne}(c_{ij})$ is the set of factors connected to c_{ij}.

Even with RBP and damping, however, sometimes the algorithm can converge to a poor biclustering configuration that violates some of the constraints. In that case, we can rerun the algorithm, not on the original factor graph, but on a simplified factor graph containing only the part of the configuration that violates the constraints. This process can be repeated, each time with a smaller graph, until the resulting biclustering configuration does not violate any constraint.

An Illustrative Example of Biclustering Using Message Passing. Figure 2 illustrates the operation of the message passing algorithm on a simple data matrix. We added very small amounts of noise into the data to break symmetry and thus avoid oscillation by ensuring that some elements are preferred over others as exemplars. It can be seen that this data matrix has a few different valid biclustering configurations, and only the biclustering conformation shown in Fig.2(a) has the smallest number of biclusters and therefore the lowest energy. At the beginning of the run, all these configurations are equally likely. During the execution of the algorithm, the data elements can be seen to switch back and forth between different candidate exemplars, and finally settle on the best configuration.

3.2 Scaling Up the Algorithm. On a data matrix consisting of n elements, the basic algorithm has a space complexity of $O(n^3)$ (there are $O(n^2)$ factors and messages, and each message requires $O(n)$ space); and it takes $O(n)$ time for each message update. So it does not scale up to settings where the data matrix contains more than a few hundred elements. In this subsection, we introduce a set of approximation techniques that enhance the scalability of the algorithm.

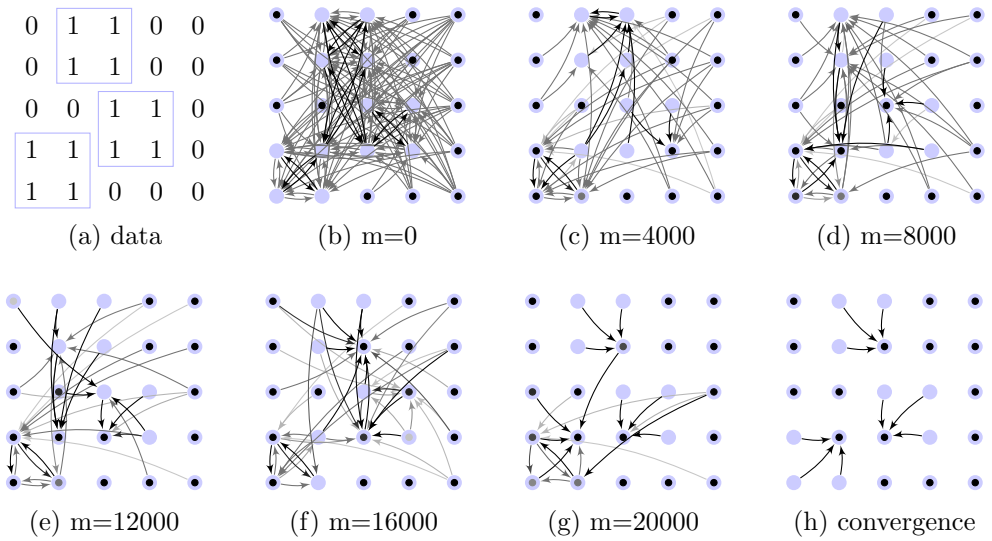
First, we reduce the complexity of ρ messages from variables to factors. Each ρ message is a vector of length n, and during an update the basic algorithm recomputes each element of this vector. However, note that when a ρ message is used in computing an α message according to Eq.3.1 or Eq.3.2, at most four values in the vector are actually used: the maximum, the second maximum, and at most two other values corresponding to the special cases. This means we only need to store at most four values in a ρ message, and each can be updated in constant time by using a cache for the variable that the message is connected to. Because the special cases can change whenever any other ρ message connected to the same factor is updated, we force all the ρ messages of the same factor to be updated at the same time.

Now we turn to the α messages from factors to variables. As described in last subsection, each α message contains only two scalars, which can be updated in constant time. However, when an α message of a variable is updated, for each of the $O(n)$ ρ messages connected to that variable, we need to recompute how much it will change if updated, and reposition it in the priority queue used by the RBP schedule. Note that according to Eq.3.3 each ρ message of a variable is the sum of all except one of the incoming α messages. Hence, we can use the change in the sum of all the incoming α messages to approximate the change in the ρ message. This forces all the ρ messages of a variable to have the same priority, which helps avoid checking each of them individually when updating a related α message.

As a result of the preceding changes, we have reduced the time and space complexity of both types of messages to $O(1)$. By exploiting the sparseness in the data matrix, we can further reduce the time and space requirement of the algorithm. First, for each element in the data matrix, usually only a small number of elements can serve as good exemplars, and the algorithm needs to only keep track of such elements. Second, if the data matrix contains a lot of background elements, we can apply some heuristics (for example, a simplified version of Bimax[20]) to quickly identify the likely background elements, which can be assigned to be their own exemplars before running the biclustering algorithm. In both cases, we can substantially reduce the number of factors as well as the size of the messages, resulting in significant savings in time and space required by the algorithm.

4 A Greedy Algorithm of Exemplar-Based Biclustering

Even with the speedups outlined above, the message passing algorithm can be too slow to be useful on very large data matrices encountered in some real-world applications. In this section we introduce a greedy algorithm of exemplar-based biclustering (EBG) that is



(a) data (b) m=0 (c) m=4000 (d) m=8000

(e) m=12000 (f) m=16000 (g) m=20000 (h) convergence

Figure 2: An illustrative example of biclustering using message passing. (a) The data matrix. The best biclustering configuration is denoted by the three boxes. (b)-(h) The exemplar preferences at different stages of the algorithm. m is the number of messages propagated. An edge from node i to node j denotes that node i prefers node j as a candidate exemplar; a dot at node i denotes that node i chooses itself as a candidate exemplar; the grayscale of the edges and dots represents the strength of the preference.

significantly faster than the message passing algorithm. The increase in speed is achieved by settling for a sub-optimal solution.

For each element a_{ij} in the data matrix, we can obtain a set of elements that prefer a_{ij} to be their exemplar, i.e., the distances between the actual and ideal values of such elements (as defined in section 2) are below a threshold if they choose a_{ij} as the exemplar. We can construct a candidate bicluster with a_{ij} as its exemplar, by collecting the rows and columns of this set of elements.

Note that in this process we do not check whether any background element prefers a_{ij} to be its exemplar, because: 1) any row or column of a true bicluster typically contains some non-background elements, and therefore is likely to be included into the candidate bicluster even if the background elements in the row or column are not checked; and 2) if the data matrix is sparse and background elements can fit into a coherent bicluster, then checking background elements can potentially introduce spurious rows or columns into the candidate bicluster.

We repeat this process for each of the elements except the background elements that are not allowed to be exemplars for any element other than themselves. The result of this process is a set of candidate biclusters, many of which can be identical or similar to one another because a true bicluster might contain multiple elements that can serve as the exemplar. Hence, a heuristic de-duplication step is carried out that removes a bicluster b_1 if b_1 is identical to or contained within another bicluster b_2 and b_1 is chosen by fewer elements as the exemplar than b_2.

Each remaining candidate bicluster needs to be shrunk. This is necessary because for any exemplar a_{ij}, an element preferring a_{ij} as its exemplar does not necessarily imply that the row and column of the element indeed belong to the bicluster that has a_{ij} as the exemplar. The shrinking of a candidate bicluster can be driven by the objective function $S(\mathbf{c})$ defined in section 2. However, this objective function $S(\mathbf{c})$ is defined over a biclustering configuration $\mathbf{c}$ and not over an individual candidate bicluster, so it cannot be directly used here. We obtain a new objective function from $S(\mathbf{c})$, first by dropping the two constraint terms (we can ensure the validity of the biclusters while shrinking them); and second by decomposing the remaining energy function $E(\mathbf{c})$ into the sum of bicluster energies $E(b)$ for each bicluster b in the configuration $\mathbf{c}$. $E(b)$ is given by:

$$E(b) = \sum_{\substack{i \in b.\text{rows} \\ j \in b.\text{columns}}} \text{CP}_{ij}(c_b)$$

where CP_{ij} is defined as in section 2.2, and c_b is the exemplar of bicluster b. So the energy $E(b)$ over an individual bicluster can now be used as the objective function to guide the shrinking of the candidate biclus-

ters. However, since now we optimize each bicluster separately, the second regularization term (which encourages smaller biclusters) is no longer balanced by the first regularization term (which encourages fewer and hence larger biclusters in $E(\mathbf{c})$, but has no effect in $E(b)$). Hence, we add a third regularization term to $E(b)$ to compensate this fact. The third regularization term is proportional to the number of non-background elements in the candidate bicluster.

$$E(b) = \sum_{\substack{i \in b.\text{rows} \\ j \in b.\text{columns}}} \text{CP}_{ij}(c_{ij}) - p_3 F(b) \tag{4.4}$$

where $F(b)$ is the number of non-background elements in b, and p_3 is a constant multiplier.

To shrink a candidate bicluster b, we greedily remove rows and/or columns to minimize $E(b)$ as defined in Eq.4.4. We do not allow the row and column of the exemplar to be removed, so as to guarantee that the bicluster contains its exemplar. After each of the candidate biclusters has been shrunk using this procedure, we perform a second round of de-duplication to eliminate any redundant bicluster, and then output the resulting biclustering.

5 Experiments

We describe results of experiments that compare the performance of the message passing algorithm and the greedy algorithm for biclustering introduced in this paper with several representative biclustering algorithms available in the literature on synthetic datasets as well as two real-world datasets.

5.1 Experiments with Synthetic Data. We compared our message passing (EBMP) and greedy (EBG) algorithms with two other coherent biclustering algorithms, Chen&Church[6] (using the BicAT software from http://www.tik.ethz.ch/sop/bicat/) and ROCC[8], on four synthetic datasets. The first two of the four datasets each contain fifteen 10×10 data matrices, and the third and fourth datasets each contain fifteen 50×50 data matrices. In each data matrix, the number, sizes, positions and coherence parameters of biclusters were all randomly generated. A 10×10 data matrix contains three biclusters on average, and the average bicluster size is 3×3; a 50×50 data matrix contains 12.5 biclusters on average, and the average bicluster size is 6×6. These biclusters constitute about one-fifth of the data matrix, while the rest are background. We only generated additively coherent biclusters, because the implementations of Chen&Church and ROCC available to us do not support discovery of multiplicatively coherent biclusters. We added small amounts of Gaussian noise (with a standard deviation being 5% of the average element value) into the matrices of the first and third datasets, and we added large amounts of Gaussian noise (with a standard deviation being 20% of the average element value) into the matrices of the second and forth datasets.

We used a variant of purity and inverse purity[2], which are widely used performance measures in the clustering literature, to measure the quality of biclustering. Suppose C is the set of biclusters found by a biclustering algorithm and L is the set of *true* biclusters in the data matrix. Suppose each bicluster is represented by the set of data elements it contains. Let C_i (L_i) be the i-th bicluster in C (L). Then we define

$$\begin{aligned} \text{Purity} &= \frac{\sum_i \max_j |C_i \cap L_j|}{\sum_i |C_i|} \\ \text{Inverse Purity} &= \frac{\sum_i \max_j |L_i \cap C_j|}{\sum_i |L_i|} \end{aligned}$$

Intuitively, purity measures the extent to which the elements assigned to each of the biclusters found by the biclustering algorithm are actually clustered together in the true biclusters; inverse purity measures the extent to which the elements belonging to each of the true biclusters are clustered together in the biclusters found by the algorithm. Both measures have the range of $[0, 1]$.

In each dataset, we used five data matrices to tune the parameters of the algorithms, and used the remaining ten data matrices for evaluating the performance of the algorithms. In the case of nondeterministic algorithms, each algorithm was run for five times on each data matrix. Because the Chen&Church algorithm cannot automatically infer the number of biclusters, we set the number of biclusters to be the same as the number of true biclusters in each run. For our message passing algorithm (EBMP), we ran the basic version on the 10×10 data matrices and the scaled-up version on the 50×50 data matrices.

Table 1 shows the results of our experiments. Both the exemplar-based message passing algorithm (EBMP) and exemplar-based greedy algorithm (EBG) introduced in this paper yield significantly better results than the Chen&Church and ROCC algorithms on the synthetic data. Inspection of the produced biclusters reveals that, even though background interference is relatively weak in the case of data matrices with additively coherent biclusters, both Chen&Church and ROCC often include too many background elements into the biclusters. In contrast, the biclusters found by our algorithms (EBMP and EBG) contain significantly fewer background elements because of the regularization terms used in our objective function.

We further observe that on the 10×10 datasets the basic version of EBMP produces almost perfect

	Chen&Church		ROCC		EBMP		EBG	
	P	IP	P	IP	P	IP	P	IP
10×10 small noise	0.1566	0.4086	0.2263	0.4262	0.9807	0.9722	0.9410	0.9009
10×10 large noise	0.1760	0.5033	0.2877	0.4262	0.8736	0.6445	0.8314	0.6489
50×50 small noise	0.0452	0.4753	0.1339	0.3639	0.7312	0.9664	0.9102	0.7859
50×50 large noise	0.0281	0.0455	0.1164	0.3848	0.7876	0.5839	0.7786	0.6786

Table 1: The experimental results on synthetic data. P and IP respectively denote purity and inverse purity. Each measurement is averaged over five runs on ten data matrices. The basic version of our message passing algorithm EBMP was run on the 10×10 data matrices and the scaled-up version of EBMP was run on the 50×50 data matrices.

biclustering results that are better than those produced by EBG; however, on the 50×50 datasets (on which we could not run the basic version of EBMP), the scaled-up version of EBMP has performance similar to that of EBG, which could be explained by the significant amount of approximations used in the scaled-up version.

It is interesting to note that, on the 10×10 datasets both Chen&Church and ROCC perform better with larger amounts of noise in the data. We speculate that this is because larger noise makes the spurious biclusters even less coherent, making it easier for Chen&Church and ROCC to exclude them from the biclustering result. In contrast, EBMP and EBG exclude most of the spurious biclusters with their built-in mechanisms regardless of the noise level, so larger noise only makes it more difficult to uncover the true biclusters.

For the running time, on a typical 50×50 data matrix, Chen&Church and our greedy algorithm EBG (both implemented in Java) finishes almost instantaneously, whereas ROCC (implemented in Matlab) runs for under a minute, and our scaled-up version of message passing algorithm EBMP (implemented in Java) runs for several minutes.

5.2 Natural Language Bigram Data. A *word bigram* is a concatenation of two words. Given a natural language corpus, we can construct a word bigram matrix, where each row or column is indexed by a word in the vocabulary, and each element of the matrix contains the frequency of the corresponding bigram (formed by the word indexing the row and the word indexing the column) in the corpus. Multiplicatively coherent biclusters of the word bigram frequency matrix have been found useful in inferring grammar rules (among other things) from a natural language corpus [1, 23]. A natural language bigram matrix is usually very sparse, with a large number of zeros. Since a row or column of zeros can fit into a bicluster while increase its multiplicative coherence, this sparsity leads to severe background interference in biclustering. Hence the bigram frequency data offers a challenging benchmark for coherent biclustering algorithms.

We used a word bigram frequency matrix constructed from the ATIS corpus of the Penn Treebank [18], which is widely used in natural language processing studies. The corpus, which is based on a vocabulary of 401 distinct words, yields a 334×341 bigram frequency matrix (after removing rows and columns with all zero entries). 98.9% of the elements in the resulting data matrix are 0s. The *correct* biclustering of the bigrams is unknown. One way to assess biclustering results is to compare a clustering of words derived from the biclustering against a reference clustering of words. Here we use a clustering of words based on the part-of-speech of words as the reference. However, it is possible to use a more refined clustering of words based on word semantics. The clustering evaluation metric is the F-measure as defined in [2]

$$F = \sum_i \frac{|L_i|}{N} \max_j \{ \frac{2|L_i \bigcap C_j|}{|L_i| + |C_j|} \}$$

where L_i is the i-th word cluster specified by the part-of-speech, C_j is the j-th word cluster derived from the biclustering result, and N is the number of words.

We compared our greedy algorithm (EBG) with Chen&Church and ROCC on the word bigram data matrix. The data matrix is too large for running our message passing algorithm (EBMP). We experimented with several settings of parameters for each of the three algorithms but found the performance to be relatively independent of the parameters (with difference in performance between parameter choices < 0.05). Here we only show the result with the best parameter setting for each algorithm (Table 2). The results show that our exemplar-based greedy algorithm (EBG) outperforms both Chen&Church and ROCC on the word bigram data. This can be explained by the fact that EBG is more robust in the presence of severe background interference arising from the large number of zeros in the data matrix. EBG also runs significantly faster on this data matrix, because EBG is designed to exploit of the sparsity of the data matrix to reduce its run time.

	F-measure
Chen&Church	0.2997 (0.0041)
ROCC	0.2909 (0.0131)
EBG	0.4446 (0)

Table 2: The experimental results on the ATIS bigram data. Each F-measure is the average over five runs. The numbers in the parentheses are the standard deviations over the five runs.

Row(s)	Column(s)
around, after	eleven, six, *noon*, seven
on, *next*	wednesday, *sunday*, saturday
discount, *business*, coach, first	*class*
coach, expensive, that, lowest, *cheapest*, on	flights, *fare*, fares
is, *serve*, have, are, serves, the	breakfast, *dinner*, lunch
in	phoenix, los, chicago, dallas, toronto, indianapolis, a, minneapolis, westchester, denver, las, washington, *new*, nashville, san, miami, milwaukee
american, united, what, *southwest*, which	*airlines*
thursday, *monday*	*afternoon*, morning, i
from, in, to, *via*	*chicago*

Table 3: Some examples of the biclusters extracted by EBG from the word bigram matrix. The words that correspond to the bicluster exemplars are italicized.

It should be noted that, if biclustering is used as a sub-routine in grammar induction, the contexts in which the bigrams occur in the sentences of the corpus need to be taken into account in generating the biclusters [23]. This requires that each element in the bigram matrix be a vector that encodes the contexts in which the bigram appears. In the experiment reported here, we have omitted the bigram context information in order to be able to use the available implementations of Chen&Church and ROCC without making the changes that would be needed to deal with vector-valued elements in the data matrix. The lack of context information partly explains the modest F-measure scores. However, as Table 3 shows, examination of the biclusters returned by EBG still reveals many meaningful word clusters.

5.3 Gene Expression Data. Analysis of gene expression data presents an instance of the biclustering problem. In a gene expression data matrix, each row corresponds to a gene, each column corresponds to a condition, and each data element contains the expression level of a genes under a conditions. Biclusters extracted from a gene expression data matrix can reveal cellular processes that typically involve only small subsets of genes that are active under subsets of conditions.

In this experiment, we follow the experimental setup of [20] as described below. We used the yeast gene expression data matrix from [12], which contains expression measurements of 2993 yeast genes under 173 conditions. Because the number of produced biclusters can vary significantly across biclustering algorithms, a greedy filtering procedure is applied so that no more than 100 largest biclusters with less than 25% overlap with each other are selected. The results are evaluated on the basis of the degree to which the biclusters are enriched with genes that share specific biological functions (as indicated by Gene Ontology (GO) annotations). More specifically, for each set of genes derived from the filtered biclustering result, we run the FuncAssociate web service [4] at five different significance levels to see if there exists a Molecular Function or Biological Process GO annotation that is overrepresented in the corresponding gene set; the biclustering result is assigned a GO enrichment score for each significance level that corresponds to the percentage of the gene sets (equivalently, biclusters) that are enriched with respect to at least one GO annotation.

We compared our greedy algorithm EBG with Chen&Church and ROCC, and also with the Bimax [20] algorithm. Bimax does not search for coherent biclusters; instead, it binarizes the data matrix and identifies constant-value biclusters. However, Bimax has been shown to perform very well on the yeast gene expression data. With each algorithm we tried several parameter settings, and the results reported correspond to the best performing parameter settings. For Chen&Church, we set the number of biclusters to be 200. After filtering the biclusters as described earlier, each of the four algorithms has 100 biclusters left (the maximal number of biclusters allowed by the filter). The resulting biclusters are compared in Figure 3 based on the GO enrichment score described above. Clearly, EBG is competitive with Chen&Church, ROCC and Bimax.

It shall be noted that although we followed the same evaluation procedure as in [20], the scores of Chen&Church and Bimax that we report are higher than the scores for the same algorithms reported in [20]. This is probably because we used a more current version of the GO enrichment evaluation software with a more

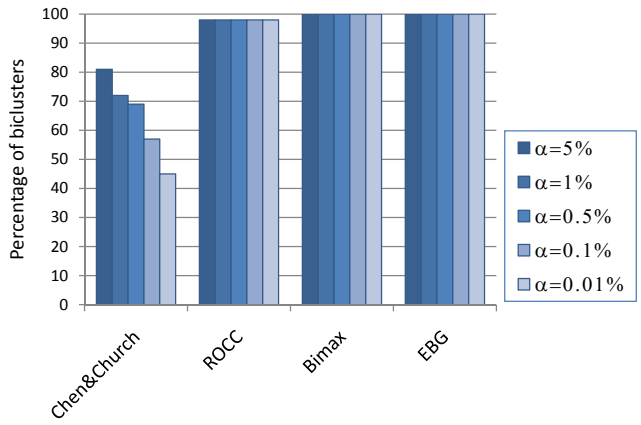


Figure 3: The experimental results on the yeast gene expression data. The y-axis is the percentage of gene clusters that are overrepresented by at least one GO annotation, with five different significance levels (α).

recent version of the Gene Ontology.

6 Conclusion and Discussion

We have introduced a novel formulation of the biclustering problem and used it to derive two novel biclustering algorithms. The first algorithm is based on loopy message passing; while the second relies on a greedy strategy that yields an algorithm that is significantly faster than the first. A distinguishing feature of these algorithms is that they identify an exemplar or a prototypical member of each bicluster. These algorithms are designed to be robust in terms of performance on sparse data, a setting in which the performance of several existing algorithms degrades because of interference from the large fraction of background (typically zero valued) elements. Our experiments with synthetic as well as real-world datasets show that the proposed algorithms are competitive with the current state-of-the-art algorithms for finding coherent biclusters.

There are some interesting directions for future work. It is of interest to explore ways to improve the scalability of our message passing algorithm. For example, our current message passing schedule incurs substantial time and space overhead. It would be interesting to experiment with some more efficient alternatives such as the one introduced by Sutton and McCallum [22]. It would also be interesting to study message passing schedules that are amenable to large-scale parallelization. Of particular interest is a hybrid algorithm that combines aspects of the greedy algorithm with aspects of the message passing algorithm to improve the quality of biclustering relative to that produced by the greedy algorithm. At any iteration of the message passing algorithm, we can read out the currently preferred exemplar(s) of each element by summing up all the incoming α messages. We can then initialize the greedy algorithm with the set of elements that prefer each exemplar. Such a hybrid algorithm might work better than the purely greedy algorithm, because the message passing procedure injects global information of the data matrix (i.e., exemplar preference of all the other elements) into the exemplar preference of each individual element. Also of interest are more comprehensive experiments that further investigate the strengths and weaknesses of alternative approaches to solving the biclustering problem.

Acknowledgement

The work of Kewei Tu was supported in part by a graduate research assistantship funded by the grant IIS 0711356 from the National Science Foundation and in part by a graduate teaching assistantship from the Department of Computer Science at Iowa State University. The work of Vasant Honavar was supported by the National Science Foundation, while working at the Foundation. Any opinions, findings, and conclusions contained in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] P. Adriaans, M. Trautwein, and M. Vervoort. Towards high speed grammar induction on large text corpora. In *SOFSEM 2000, LNCS 1963*, 2000.

[2] Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Inf. Retr.*, 12(4):461–486, 2009.

[3] Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. *J. Mach. Learn. Res.*, 8:1919–1986, 2007.

[4] Gabriel F. Berriz, John E. Beaver, Can Cenik, Murat Tasan, and Frederick P. Roth. Next generation software for functional trend analysis. *Bioinformatics*, 25(22):3043–3044, 2009.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning.*, chapter 8.4, pages 393–422. Springer, 2006.

[6] Yizong Cheng and George M. Church. Biclustering of expression data. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB-00)*, pages 93–103, Menlo Park, CA, August 16–23 2000. AAAI Press.

[7] Hyuk Cho, Inderjit S. Dhillon, Yuqiang Guan, and Suvrit Sra. Minimum sum-squared residue co-clustering

of gene expression data. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, 2004.

[8] Meghana Deodhar, Gunjan Gupta, Joydeep Ghosh, Hyuk Cho, and Inderjit S. Dhillon. A scalable framework for discovering coherent co-clusters in noisy data. In *ICML*, page 31, 2009.

[9] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, pages 269–274, 2001.

[10] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *in Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI*, 2006.

[11] Brendan J. Frey and Delbert Dueck. Mixture modeling by affinity propagation. In *NIPS*, volume 18, 2005.

[12] AP Gasch, PT Spellman, CM Kao, O Carmel-Harel, MB Eisen, G Storz, D Botstein, and PO Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Mol Biol Cell*, 11(12):4241–57, Dec 2000.

[13] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. *Proc Natl Acad Sci U S A*, 97(22):12079–12084, October 2000.

[14] Hartigan JA. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337), 1972.

[15] Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, 2006.

[16] Laura Lazzeroni and Art Owen. Plaid models for gene expression data. *Statistica Sinica*, 12:61–86, 2000.

[17] S. C. Madeira and A. L. Oliviera. Biclustering algorithms for biological data analysis: a survey. *IEEE Trans. on Comp. Biol. and Bioinformatics*, 1(1):24–45, 2004.

[18] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330, 1993.

[19] Edward Meeds, Zoubin Ghahramani, Radford M. Neal, and Sam T. Roweis. Modeling dyadic data with binary latent factors. In *NIPS*, pages 977–984, 2006.

[20] Amela Prelić, Stefan Bleuler, Philip Zimmermann, Anja Wille, Peter Bühlmann, Wilhelm Gruissem, Lars Hennig, Lothar Thiele, and Eckart Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, 2006.

[21] Eran Segal, Alexis Battle, and Daphne Koller. Decomposing gene expression into cellular processes. In *Pacific Symposium on Biocomputing*, pages 89–100, 2003.

[22] Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[23] Kewei Tu and Vasant Honavar. Unsupervised learning of probabilistic context-free grammar using iterative biclustering. In *Proceedings of 9th International Colloquium on Grammatical Inference (ICGI 2008), LNCS 5278*, September 2008.

[24] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD Conference*, pages 394–405, 2002.

[25] Jiong Yang, Wei Wang, Haixun Wang, and Philip S. Yu. δ-clusters: Capturing subspace correlation in a large data set. In *ICDE*, pages 517–528. IEEE Computer Society, 2002.