Biclustering Using Message Passing

Luke O’Connor  
Bioinformatics and Integrative Genomics  
Harvard University  
Cambridge, MA 02138  
loconnor@g.harvard.edu

Soheil Feizi  
Electrical Engineering and Computer Science  
Massachusetts Institute of Technology  
Cambridge, MA 02139  
sfeizi@mit.edu

Abstract

Biclustering is the analog of clustering on a bipartite graph. Existent methods infer biclusters through local search strategies that find one cluster at a time; a common technique is to update the row memberships based on the current column memberships, and vice versa. We propose a biclustering algorithm that maximizes a global objective function using message passing. Our objective function closely approximates a general likelihood function, separating a cluster size penalty term into row- and column-count penalties. Because we use a global optimization framework, our approach excels at resolving the overlaps between biclusters, which are important features of biclusters in practice. Moreover, Expectation-Maximization can be used to learn the model parameters if they are unknown. In simulations, we find that our method outperforms two of the best existing biclustering algorithms, ISA and LAS, when the planted clusters overlap. Applied to three gene expression datasets, our method finds coregulated gene clusters that have high quality in terms of cluster size and density.

1 Introduction

The term biclustering has been used to describe several distinct problems variants. In this paper, In this paper, we consider the problem of biclustering as a bipartite analogue of clustering: Given an $N \times M$ matrix, a bicluster is a subset of rows that are heavily connected to a subset of columns. In this framework, biclustering methods are data mining techniques allowing simultaneous clustering of the rows and columns of a matrix. We suppose there are two possible distributions for edge weights in the bipartite graph: a within-cluster distribution and a background distribution. Unlike in the traditional clustering problem, in our setup, biclusters may overlap, and a node may not belong to any cluster. We emphasize the distinction between biclustering and the bipartite analog of graph partitioning, which might be called bipartitioning.

Biclustering has several noteworthy applications. It has been used to find modules of coregulated genes using microarray gene expression data [1] and to predict tumor phenotypes from their genotypes [2]. It has been used for document classification, clustering both documents and related words simultaneously [3]. In all of these applications, biclusters are expected to overlap with each other, and these overlaps themselves are often of interest (e.g., if one wishes to explore the relationships between document topics).

The biclustering problem is NP-hard (see Proposition 1). However, owing to its practical importance, several heuristic methods using local search strategies have been developed. A popular approach is to search for one bicluster at a time by iteratively assigning rows to a bicluster based on the columns, and vice versa. Two algorithms based on this approach are ISA [4] and LAS [5]. Another approach is an exhaustive search for complete bicliques used by Bimax [6]. This approach fragments large noisy clusters into small complete ones. SAMBA [7] uses a heuristic combinatorial search for locally optimal biclusters, motivated by an exhaustive search algorithm that is exponential.
in the maximum degree of the nodes. For more details about existent biclustering algorithms, and
performance comparisons, see references [6] and [8]. Existent biclustering methods have two major
shortcomings: first, they apply a local optimality criterion to each bicluster individually. Because a
collection of locally optimal biclusters might not be globally optimal, these local methods struggle
to resolve overlapping clusters, which arise frequently in many applications. Second, the lack of
a well-defined global objective function precludes an analytical characterization of their expected
results.

Global optimization methods have been developed for problems closely related to biclustering, in-
cluding clustering. Unlike most biclustering problem formulations, these are mostly partitioning
problems: each node is assigned to one cluster or category. Major recent progress has been made in
the development of spectral clustering methods (see references [9] and [10]) and message-passing
algorithms (see [11], [12] and [13]). In particular, Affinity Propagation [12] maximizes the sum of
similarities to one central exemplar instead of overall cluster density. Reference [14] uses variational
expectation-maximization to fit the latent block model, which is a binary model in which each row
or column is assigned to a row or column cluster, and the probability of an edge is dictated by the
respective cluster memberships. Row and column clusters that are not paired to form biclusters.

In this paper, we propose a message-passing algorithm that searches for a globally optimal col-
lection of possibly overlapping biclusters. Our method maximizes a likelihood function using an
approximation that separates a cluster-size penalty term into a row-count penalty and a column-
count penalty. This decoupling enables the messages of the max-sum algorithm to be computed
efficiently, effectively breaking an intractable optimization into a pair of tractable ones that can be
solved in nearly linear time. When the underlying model parameters are unknown, they can be
learned using an expectation-maximization approach.

Our approach has several advantages over existing biclustering algorithms: the objective function
of our biclustering method has the flexibility to handle diverse statistical models; the max-sum al-
gorithm is a more robust optimization strategy than commonly used iterative approaches; and in
particular, our global optimization technique excels at resolving overlapping biclusters. In simula-
tions, our method outperforms two of the best existing biclustering algorithms, ISA and LAS, when
the planted clusters overlap. Applied to three gene expression datasets, our method found biclusters
of high quality in terms of cluster size and density.

2 Methods

2.1 Problem statement

Let $G = (V, W, E)$ be a weighted bipartite graph, with vertices $V = (1, \ldots, N)$ and $W = (1, \ldots, M)$,
connected by edges with non-negative weights: $E : V \times W \rightarrow [0, \infty)$. Let $V_1, \ldots, V_K \subset V$ and
$W_1, \ldots, W_K \subset W$. Let $(V_k, W_k) = \{(i, j) : i \in V_k, j \in W_k\}$ be a bicluster: Graph edge weights
e_{ij} are drawn independently from either a within-cluster distribution or a background distribution
depending on whether, for some $k$, $i \in V_k$ and $j \in W_k$. In this paper, we assume that the within-
cluster and background distributions are homogenous. However, our formulation can be extended
to a general case in which the distributions are row- or column-dependent.

Let $c_{ij}^k$ be the indicator for $i \in V_k$ and $j \in W_k$. Let $c_{ij} \triangleq \min(1, \sum_k c_{ij}^k)$ and let $c \triangleq (c_{ij}^k)$.

**Definition 1 (Biclustering Problem).** Let $G = (V, W, E)$ be a bipartite graph with biclusters
$(V_1, W_1), \ldots, (V_K, W_K)$, within-cluster distribution $f_1$ and background distribution $f_0$. The problem
is to find the maximum likelihood cluster assignments (up to reordering):

$$
\hat{c} = \arg \max_c \sum_{i,j} c_{ij} \log \frac{f_1(c_{ij})}{f_0(c_{ij})}, \tag{1}
$$

$$
c_{ij}^k = c_{rs}^k = 1 \Rightarrow c_{ra}^k = c_{sj}^k = 1, \quad \forall i, r \in V, \forall j, s \in W.
$$

Figure 1 demonstrates the problem qualitatively for an unweighted bipartite graph. In general, the
combinatorial nature of a biclustering problem makes it computationally challenging.

**Proposition 1.** The clique problem can be reduced to the maximum likelihood problem of Definition
(1). Thus, the biclustering problem is NP-hard.
Figure 1: Biclustering is the analogue of clustering on a bipartite graph. (a) Biclustering allows nodes to be reordered in a manner that reveals modular structures in the bipartite graph. (b) The rows and columns of an adjacency matrix are similarly biclustered and reordered.

Proof. Proof is provided in Supplementary Note 1.

2.2 BCMP objective function

In this section, we introduce the global objective function considered in the proposed biclustering algorithm called Biclustering using Message Passing (BCMP). This objective function approximates the likelihood function of Definition 1. Let \( l_{ij} = \log \frac{f_1(e_{ij})}{f_0(e_{ij})} \) be the log-likelihood ratio score of tuple \((i, j)\). Thus, the likelihood function of Definition 1 can be written as \( \sum_{ij} c_{ij} l_{ij} \). If there were no consistency constraints in the Optimization (1), an optimal maximum likelihood biclustering solution would be to set \( c_{ij} = 1 \) for all tuples with positive \( l_{ij} \). Our key idea is to enforce the consistency constraints by introducing a cluster-size penalty function and shifting the log-likelihood ratios \( l_{ij} \) to recoup this penalty. Let \( N_k \) and \( M_k \) be the number of rows and columns, respectively, assigned to cluster \( k \). We have,

\[
\sum_{(i,j)} c_{ij} l_{ij} \approx \sum_{(i,j)} c_{ij} \max(0, l_{ij} + \delta) - \delta \sum_{(i,j)} c_{ij} \\
\approx \sum_{(i,j)} c_{ij} \max(0, l_{ij} + \delta) + \delta \sum_{(i,j)} \max(0, -1 + \sum_k c_{ij}^k) - \delta \sum_k N_k M_k \\
\approx \sum_{(i,j)} c_{ij} \max(0, l_{ij} + \delta) + \delta \sum_{(i,j)} \max(0, -1 + \sum_k c_{ij}^k) - \frac{\delta}{2} \sum_k r_k N_k^2 + r_k^{-1} M_k^2.
\]

(2)

The approximation \((a)\) holds when \( \delta \) is large enough that thresholding \( l_{ij} \) at \(-\delta\) has little effect on the resulting objective function. In equation \((b)\), we have expressed the second term of \((a)\) in terms of a cluster size penalty \(-\delta N_k M_k\), and we have added back a term corresponding to the overlap between clusters. Because a cluster-size penalty function of the form \( N_k M_k \) leads to an intractable optimization in the max-sum framework, we approximate it using a decoupling approximation \((c)\) where \( r_k \) is a cluster shape parameter:

\[
2N_k M_k \approx r_k N_k^2 + r_k^{-1} M_k^2,
\]

(3)

when \( r_k \approx M_k/N_k \). The cluster-shape parameter can be iteratively tuned to fit the estimated biclusters.

Following equation (2), the BCMP objective function can be separated into three terms as follows:
Definition 2. The binary biclustering model is a generative model for a binary or unweighted bipartite graph: 3.3). Now, we analyze BCMP over the following model for a binary or unweighted bipartite graph:

\[
F(c) = \sum_{i,j} \tau_{ij} + \sum_k \eta_k + \sum_k \mu_k,
\]

\[
\begin{align*}
\tau_{ij} &= \ell_{ij} \min(1, \sum_k \epsilon_{ij}^k) + \delta \max(0, \sum_k \epsilon_{ij}^k - 1) \quad \forall (i,j) \in V \times W, \\
\eta_k &= -\frac{\delta}{2} r_k N_k^2 \quad \forall 1 \leq k \leq K, \\
\mu_k &= -\frac{\delta}{2} r_k^{-1} M_k^2 \quad \forall 1 \leq k \leq K
\end{align*}
\]

Here \(\tau_{ij}\), the tuple function, encourages heavier edges of the bipartite graph to be clustered. Its second term compensates for the fact that when biclusters overlap, the cluster-size penalty functions double-count the overlapping regions. \(\ell_{ij} \triangleq \max(0, l_{ij} - \delta)\) is the shifted log-likelihood ratio for observed edge weight \(e_{ij}\). \(\eta_k\) and \(\mu_k\) penalize the number of rows and columns of cluster \(k\), \(N_k\) and \(M_k\), respectively. Note that by introducing a penalty for each nonempty cluster, the number of clusters can be learned, and finding weak, spurious clusters can be avoided (see Supplementary Note 3.3).

Now, we analyze BCMP over the following model for a binary or unweighted bipartite graph:

**Definition 2.** The binary biclustering model is a generative model for \(N \times M\) bipartite graph \((V, W, E)\) with \(K\) biclusters distributed by uniform sampling with replacement, allowing for overlapping clusters. Within a bicluster, edges are drawn independently with probability \(p\), and outside of a bicluster, they are drawn independently with probability \(q < p\).

In the following, we assume that \(p, q\), and \(K\) are given. We discuss the case that the model parameters are unknown in Section 2.4. The following proposition shows that optimizing the BCMP objective function solves the problem of Definition 1 in the case of the binary model:

**Proposition 2.** Let \((e_{ij})\) be a matrix generated by the binary model described in Definition 2. Suppose \(p, q\) and \(K\) are given. Suppose the maximum likelihood assignment of edges to biclusters, \(\arg \max P(\text{data}|c)\), is unique up to reordering. Let \(r_k = M_k^l / N_k^l\) be the cluster shape ratio for the \(k\)-th maximum likelihood cluster. Then, by using these values of \(r_k\), setting \(\ell_{ij} = e_{ij}\), for all \((i, j)\), with cluster size penalty

\[
\delta = -\frac{\log(1 - p) - \log(1 - q)}{2 \log(p/q)}\]

we have,

\[
\arg \max_c P(\text{data}|c) = \arg \max_c (F(c)).
\]

**Proof.** The proof follows the derivation of equation (2). It is presented in Supplementary Note 2. \(\square\)

**Remark 1.** In the special case when \(q = 1 - p \in (0,1/2)\), according to equation (6), we have \(\delta = 1/2\). This is suggested as a reasonable initial value to choose when the true values of \(p\) and \(q\) are unknown; see Section 2.4 for a discussion of learning the model parameters.

The assumption that \(r_k = N_k^l / M_k^l\) may seem rather strong. However, it is essential as it justifies the decoupling equation (3) that enables a linear-time algorithm. In practice, if the initial choice of \(r_k\) is close enough to the actual ratio that a cluster is detected corresponding to the real cluster, \(r_k\) can be tuned to find the true value by iteratively updating it to fit the estimated bicluster. This iterative strategy works well in our simulations. For more details about automatically tuning the parameter \(r_k\), see Supplementary Note 3.1.

In a more general statistical setting, log-likelihood ratios \(l_{ij}\) may be unbounded below, and the first step \((a)\) of derivation (2) is an approximation; setting \(\delta\) arbitrarily large will eventually lead to instability in the message updates.
2.3 Biclustering Using Message Passing

In this section, we use the max-sum algorithm to optimize the objective function of equation (4). For a review of the max-sum message update rules, see Supplementary Note 4. There are \( NM \) function nodes for the functions \( \tau_{ij} \), \( K \) function nodes for the functions \( \eta_k \), and \( K \) function nodes for the functions \( \mu_k \). There are \( NMK \) binary variables, each attached to three function nodes: \( c^k_{ij} \) is attached to \( \tau_{ij}, \eta_k, \) and \( \mu_k \) (see Supplementary Figure 1). The incoming messages from these function nodes are named \( t_{ij}^k, n_{ij}^k, \) and \( m_{ij}^k \), respectively. In the following, we describe messages for \( c^k_{ij} = c_{12}^k \); other messages can be computed similarly.

First, we compute \( t_{12}^1 \):

\[
\begin{align*}
\ell_{12}^1(x) &= \max_{c_{12}^1, \ldots, c_{K}^1} \left[ \tau_{12}(x, c_{12}^1, \ldots, c_{K}^1) + \sum_{k \neq 1} n_{12}^k(c_{12}^k) + n_{12}^k(c_{12}^k) \right] \quad (8) \\
&= \max_{c_{12}^1, \ldots, c_{K}^1} \left[ \ell_{12} \min (1, \sum_{k} c_{ij}^k) + \delta \max (0, \sum_{k} c_{ij}^k - 1) + \sum_{k \neq 1} c_{ij}^k(n_{12}^k + n_{12}^k) \right] + d_1 \\
\end{align*}
\]

where \( d_1 = \sum_{k \neq 1} n_{12}^k(0) + n_{12}^k(0) \) is a constant. Equality (a) comes from the definition of messages according to equation (6) in the Supplement. Equality (b) uses the definition of \( \tau_{12} \) of equation (5) and the definition of the scalar message of equation (8) in the Supplement. We can further simplify \( t_{12}^1 \) as follows:

\[
\begin{align*}
\ell_{12}^1(1) - d_1 &\leq \ell_{12} + \sum_{k \neq 1} \max (0, \delta + m_{12}^k + n_{12}^k), \\
\ell_{12}^1(0) - d_1 &\leq \ell_{12} - \delta + \sum_{k \neq 1} \max (0, \delta + m_{12}^k + n_{12}^k), \quad \text{if } \exists k, n_{12}^k + m_{12}^k + \delta > 0, \\
\ell_{12}^1(0) - d_1 &\leq \max (0, \ell_{12} + \max k \neq 1 (m_{12}^k + n_{12}^k)), \quad \text{otherwise}.
\end{align*}
\]

If \( c_{ij}^1 = 1 \), we have \( \min (1, \sum_{k} c_{ij}^k) = 1 \), and \( \max (0, \sum_{k} c_{ij}^k - 1) = \sum_{k \neq 1} c_{ij}^k \). These lead to equality (c). A similar argument can be made if \( c_{ij}^1 = 0 \) but there exists a \( k \) such that \( n_{ij}^k + m_{ij}^k + \delta > 0 \). This leads to equality (d). If \( c_{ij}^k = 0 \) and there is no \( k \) such that \( n_{ij}^k + m_{ij}^k + \delta > 0 \), we can compute the increase obtained by letting \( c_{ij}^1 = 1 \) (i.e., \( \ell_{12} \)) with the penalty (i.e., \( m_{ij}^k + n_{ij}^k \)), for the best \( k \). This leads to equality (e).

**Remark 2.** Computation of \( t_{ij}^1, \ldots, t_{ij}^k \) using equality (d) costs \( O(K) \), and not \( O(K^2) \), as the summation need only be computed once.

Messages \( m_{12}^1 \) and \( n_{12}^1 \) are computed as follows:

\[
\begin{align*}
m_{12}^1(x) &= \max_{c_{12}^1 \leq c_{ij}^1 = 0} \left[ \eta_1(c_{ij}^1) + \sum_{(i,j) \neq (1,2)} t_{ij}^1(c_{ij}^1) + n_{ij}^1(c_{ij}^1) \right], \\
n_{12}^1(x) &= \max_{c_{12}^1 \leq c_{ij}^1 = 0} \left[ \eta_1(c_{ij}^1) + \sum_{(i,j) \neq (1,2)} t_{ij}^1(c_{ij}^1) + m_{ij}^1(c_{ij}^1) \right],
\end{align*}
\]

where \( c_{ij}^1 = \{ c_{ij}^k : i \in V, j \in W \} \). To compute \( n_{12}^1 \) in constant time, we perform a preliminary optimization, ignoring the effect of edge \((1,2)\):

\[
\text{argmax } c_{ij}^1 \frac{-\delta}{2} N_i^2 + \sum_{(i,j)} t_{ij}^1(c_{ij}^1) + m_{ij}^1(c_{ij}^1).
\]

Let \( s_i = \sum_{j=1}^M \max (0, m_{ij}^1 + t_{ij}^1) \) be the sum of positive incoming messages of row \( i \). The function \( \eta_1 \) penalizes the number of rows containing some nonzero \( c_{ij}^1 \): if any message along that row is included, there is no additional penalty for including every positive message along that row. Thus, optimization (11) is computed by deciding which rows to include. This can be done efficiently through sorting: we sort row sums \( s_1, \ldots, s_N \) at a cost of \( O(N \log N) \). Then we proceed from largest to smallest, including row \((N + 1 - i)\) if the marginal penalty \( \frac{\delta}{2} (i^2 - (i - 1)^2) = \frac{\delta}{2} (2i - 1) \) is less than \( s_{(N + 1 - i)} \). After solving optimization (11), the messages \( n_{12}^1, \ldots, n_{N2}^1 \) can be computed in linear time, as we explain in Supplementary Note 5.

**Remark 3.** Computation of \( n_{ij}^k \) through sorting costs \( O(N \log N) \).

**Proposition 3** (Computational Complexy of BCMP). The computational complexity of BCMP over a bipartite graph with \( N \) rows, \( M \) columns, and \( K \) clusters is \( O(K(N + \log M)(M + \log N)) \).
Proof. For each iteration, there are $NM$ messages $t_{ij}$ to be computed at cost $O(K)$ each. Before computing $(n^{ik}_j)$, there are $K$ sorting steps at a cost of $O(M \log M)$, after which each message may be computed in constant time. Likewise, there are $K$ sorting steps at a cost of $O(N \log N)$ each before computing $(m^{ik}_j)$. \hfill \Box

We provide an empirical runtime example of the algorithm in Supplementary Figure 3.

### 2.4 Parameter learning using Expectation-Maximization

In the BCMP objective function described in Section 2.2, the parameters of the generative model were used to compute the log-likelihood ratios $(l_{ij})$. In practice, however, these parameters may be unknown. Expectation-Maximization (EM) can be used to estimate these parameters. The use of EM in this setting is slightly unorthodox, as we estimate the hidden labels (cluster assignments) in the M step instead of the E step. However, the distinction between parameters and labels is not intrinsic in the definition of EM [15] and the true ML solution is still guaranteed to be a fixed point of the iterative process. Note that it is possible that the EM iterative procedure leads to a locally optimal solution and therefore it is recommended to use several random re-initializations for the method.

The EM algorithm has three steps:

- **Initialization**: We choose initial values for the underlying model parameters $\theta$ and compute the log-likelihood ratios $(l_{ij})$ based on these values, denoting by $F_0$ the initial objective function.
- **M step**: We run BCMP to maximize the objective $F_i(c)$. We denote the estimated cluster assignments by by $\hat{c}_i$.
- **E step**: We compute the expected-log-likelihood function as follows:

$$F_{i+1}(c) = E_{\theta}[\log P((e_{ij})|\theta)]= \sum_{i,j} E_{\theta}[\log P(e_{ij}|\theta)|c = \hat{c}_i]. \quad (12)$$

Conveniently, the expected-likelihood function takes the same form as the original likelihood function, with an input matrix of expected log-likelihood ratios. These can be computed efficiently if conjugate priors are available for the parameters. Therefore, BCMP can be used to maximize $F_{i+1}$. The algorithm terminates upon failure to improve the estimated likelihood $F_i(\hat{c}_i)$.

For a discussion of the application of EM to the binary and Gaussian models, see Supplementary Note 6. In the case of the binary model, we use uniform Beta distributions as conjugate priors for $p$ and $q$, and in the case of the Gaussian model, we use inverse-gamma-normal distributions as the priors for the variances and means. Even when convenient priors are not available, EM is still tractable as long as one can sample from the posterior distributions.

### 3 Evaluation results

We compared the performance of our biclustering algorithm with two methods, ISA and LAS, in simulations and in real gene expression datasets (Supplementary Note 8). ISA was chosen because it performed well in comparison studies [6] [8], and LAS was chosen because it outperformed ISA in preliminary simulations. Both ISA and LAS search for biclusters using iterative refinement. ISA assigns rows iteratively to clusters fractionally in proportion to the sum of their entries over columns. It repeats the same for column-cluster assignments, and this process is iterated until convergence. LAS uses a similar greedy iterative search without fractional memberships, and it masks already-detected clusters by mean subtraction.

In our simulations, we generate simulated bipartite graphs of size 100x100. We planted (possibly overlapping) biclusters as full blocks with two noise models:

- **Bernoulli noise**: we drew edges according to the binary model of Definition 2 with varying noise level $q = 1 - p$. 

For each of these cases, we ran simulations on three setups (see Figure 2):

- **Non-overlapping clusters**: three non-overlapping biclusters were planted in a $100 \times 100$ matrix with sizes $20 \times 20$, $15 \times 20$, and $15 \times 10$. We varied the noise level.
- **Overlapping clusters with fixed overlap**: Three overlapping biclusters with fixed overlaps were planted in a $100 \times 100$ matrix with sizes $20 \times 20$, $20 \times 10$, and $10 \times 30$. We varied the noise level.
- **Overlapping clusters with variable overlap**: we planted two $30 \times 30$ biclusters in a $100 \times 100$ matrix with variable amount of overlap between them, where the amount of overlap is defined as the fraction of rows and columns shared between the two clusters. We used Bernoulli noise level $q = 1 - p = 0.15$, and Gaussian noise level $\sigma = 0.7$.

The methods used have some parameters to set. Pseudocode for BCMP is presented in Supplementary Note 10. Here are the parameters that we used to run each method:

- **BCMP method with underlying parameters given**: We computed the input matrix of shifted log-likelihood ratios following the discussion in Section 2.2. The number of biclusters $K$ was given. We initialized the cluster-shape parameters $r_k$ at 1 and updated them as discussed in Supplementary Note 3.1. In the case of Bernoulli noise, following Proposition 2 and Remark 1, we set $\delta_{ij} = e_{ij}$ and $\frac{1}{\delta} = 1/4$. In the case of Gaussian noise, we chose a threshold $\delta$ to maximize the unthresholded likelihood (see Supplementary Note 3.2).
- **BCMP - EM method**: Instead of taking the underlying model parameters as given, we estimated them using the procedure described in Section 2.4 and Supplementary Note 6.
We used identical, uninformative priors on the parameters of the within-cluster and null distributions.

- **ISA method:** We used the same threshold ranges for both rows and columns, attempting to find best-performing threshold values for each noise level. These values were mostly around 1.5 for both noise types and for all three dataset types. We found positive biclusters, and used 20 reinitializations. Out of these 20 runs, we selected the best-performing run.

- **LAS method:** There were no parameters to set. Since $K$ was given, we selected the first $K$ biclusters discovered by LAS, which marginally increased its performance.

Evaluation results of both noise models and non-overlapping and overlapping biclusters are shown in Figure 2. In the non-overlapping case, BCMP and LAS performed similarly well, better than ISA. Both of these methods made few or no errors up until noise levels $q = 0.2$ and $\sigma = .6$ in Bernoulli and Gaussian cases, respectively. When the parameters had to be estimated using EM, BCMP performed worse for higher levels of Gaussian noise but well otherwise. ISA outperformed BCMP and LAS at very high levels of Bernoulli noise; at such a high noise level, however, the results of all three algorithms are comparable to a random guess.

In the presence of overlap between biclusters, BCMP outperformed both ISA and LAS except at very high noise levels. Whereas LAS and ISA struggled to resolve these clusters even in the absence of noise, BCMP made few or no errors up until noise levels $q = 0.2$ and $\sigma = .6$ in Bernoulli and Gaussian cases, respectively. Notably, the overlapping clusters were more asymmetrical, demonstrating the robustness of the strategy of iteratively tuning $r_k$ in our method. In simulations with variable overlaps between biclusters, for both noise models, BCMP outperformed LAS significantly, while the results for the ISA method were very poor (data not shown). These results demonstrate that BCMP excels at inferring overlapping biclusters.

### 4 Discussion and future directions

In this paper, we have proposed a new biclustering technique called Biclustering Using Message Passing that, unlike existent methods, infers a globally optimal collection of biclusters rather than a collection of locally optimal ones. This distinction is especially relevant in the presence of overlapping clusters, which are common in most applications. Such overlaps can be of importance if one is interested in the relationships among biclusters. We showed through simulations that our proposed method outperforms two popular existent methods, ISA and LAS, in both Bernoulli and Gaussian noise models, when the planted biclusters were overlapping. We also found that BCMP performed well when applied to gene expression datasets.

Biclustering is a problem that arises naturally in many applications. Often, a natural statistical model for the data is available; for example, a Poisson model can be used for document classification (see Supplementary Note 9). Even when no such statistical model will be available, BCMP can be used to maximize a heuristic objective function such as the modularity function [17]. This heuristic is preferable to clustering the original adjacency matrix when the degrees of the nodes vary widely; see Supplementary Note 7.

The same optimization strategy used in this paper for biclustering can also be applied to perform clustering, generalizing the graph-partitioning problem by allowing nodes to be in zero or several clusters. We believe that the flexibility of our framework to fit various statistical and heuristic models will allow BCMP to be used in diverse clustering and biclustering applications.

### Acknowledgments

We would like to thank Professor Manolis Kellis and Professor Muriel Médard for their advice and support. We would like to thank the Harvard Division of Medical Sciences for supporting this project.
References

Supplementary Notes

Biclustering Using Message Passing

Luke O’Connor
Bioinformatics and Integrative Genomics
Harvard University
Cambridge, MA 02138
loconnor@g.harvard.edu

Soheil Feizi
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
sfeizi@mit.edu

1 Proof of Proposition 1

In the case $K = 1$ the problem (1) is equivalent to the problem of finding a maximum weight biclique on the graph $G = (U, V, (l_{ij}))$ with edge weights $l_{ij} = \log \frac{f_1(e_{ij})}{f_0(e_{ij})}$. This problem contains as a special case the problem of finding a maximum size complete clique on a graph $(V, E)$: let $U = V$ and $l_{ii} = 1$. For $i \neq j$, let $l_{ij} = l_{ji} = 0$ if $(i, j) \in E$ and $-\infty$ otherwise. Suppose the maximal clique on $(V, E)$ is $W$. Then the biclique $(W, W')$ has weight $|W|$. For every other biclique $(U', V')$, if its weight is not $-\infty$, then $U' \cap V'$ must correspond to a complete clique on $(V, E)$; its weight is $|U' \cap V'| \leq |W|$, thus $W$ is a solution to (1) on $G$, and the problem (1) is NP-hard.

2 Proof of Proposition 2

This proof is essentially a pedantic version of the derivation of equation (2) in the main text. Given a putative set of biclusters $(V_1, W_1), \ldots, (V_K, W_K)$, let $c = (c_{ij})$ be the $N \times M$ matrix with ones where $i, j$ are in the same bicluster and zeroes elsewhere. If the data is $E = (e_{ij})$, with ones where edges are observed, then the probability of the data given these biclusters is just the product of the probabilities of the tuples,

$$P(E|I) = \Pi_{i \leq N, j \leq M} t_{ij}$$

where,

$$t_{ij} = \begin{cases} p & e_{ij} = c_{ij} = 1 \\ q & e_{ij} = 1, c_{ij} = 0 \\ 1 - p & e_{ij} = 0, c_{ij} = 1 \\ 1 - q & e_{ij} = c_{ij} = 0 \end{cases}$$ (1)

The maximum likelihood set of biclusters maximize this expression. Equivalently, we maximize the sum, over all tuples $(i, j)$ placed in a bicluster, of the log-likelihood ratios, $\log(p/q)$ if $e_{ij} = 1$ and $\log((1 - p)/(1 - q))$ otherwise.

Let the input matrix be $(\ell_{ij}) = (e_{ij})$, and let the cluster-size penalty coefficient be

$$\delta = -\frac{\log(\frac{1-p}{1-q})}{2 \log(\frac{p(1-q)}{q(1-p)})}.$$ (2)

Let $c_{ij}$ be the indicator for row-column pair $ij$ being in cluster $k$ for some $k$. First,
\[ F(c) = \sum_{ij} c_{ij}(\ell_{ij} + (\sum_k c_{ij}^k - 1) \cdot \delta) - \frac{\delta}{2} \sum_k r_k N_k^2 + r_k^{-1} M_k^2 \]

\[ = \sum_{ij} c_{ij}(\ell_{ij} + (\sum_k c_{ij}^k - 1) \cdot \delta) - \delta(\sum_k N_k M_k + \frac{1}{2}(r_k N_k - r_k^{-1} M_k)^2). \]

Now, if each \( r_k = M_k/N_k \), then the terms \( \frac{\delta}{2}(r_k N_k - r_k^{-1} M_k)^2 \) drop out; otherwise, there is an additional penalty. The additional penalty cannot hurt, since it is zero when \( c = c' \) and nonpositive otherwise. Notice that if a tuple is assigned to more than one cluster, the term \( (\sum_k c_{ij}^k - 1) \frac{\delta}{2} \) cancels the fact that it is counted in two cluster-size penalty functions. By dropping the cluster shape term we get

\[ F(c) \geq \sum_{ij} c_{ij}(\ell_{ij} + (\sum_k c_{ij}^k - 1) \cdot \delta) - \delta \sum_k N_k M_k \]

\[ = \sum_{ij} c_{ij}(\ell_{ij} + \delta) \tag{3} \]

\[ \propto |\{(i, j) : \text{tuple } ij \text{ is placed in a bicluster and } e_{ij} = 1\}| \cdot \log\left(\frac{P}{q}\right) \tag{4} \]

\[ + |\{(i, j) : \text{tuple } ij \text{ is placed in a bicluster and } e_{ij} = 0\}| \cdot \log\left(\frac{1 - P}{1 - q}\right) \]

\[ = \log(P(\text{data}|c)) + \text{const} \]

with equality when each \( r_k = N_k/M_k \). Equality (3) follows from the fact that \( \sum_k N_k M_k \) is the number of clustered tuples plus the amount of overlap between the clusters, and this overlap term cancels the term \( \sum_{ij} c_{ij}(\sum_k c_{ij}^k - 1) \). Step (4) follows from the choice of \( \ell_{ij} \). This completes the proof of 2.3.

3 Automatic parameter tuning

3.1 Automatically tuning \( r_k \)

Automatically tuning \( r_k \) to fit the observed biclusters allows BCMP to find differently-shaped clusters. After each message update, it updates \( r_{new} = \sqrt{M_k/N_k} \cdot \sqrt{r_{old}} \).

This strategy works well in practice. It can err by finding two biclusters with shapes closer to the original choice of \( r_k \) instead of a single, differently shaped cluster; to catch such errors, it is helpful to visualize the biclustering results or to check that no two clusters have mostly the same rows or columns.

3.2 Automatically tuning \( \delta \)

The offset parameter \( \delta \) is tuned by maximizing the unthresholded likelihood function. BCMP is run for several values of \( \delta \), and for each solution, the likelihood is computed. The solution with the greatest likelihood is kept.

3.3 Automatically detecting the number of clusters

When the true number of clusters is unknown, a penalty for each nonempty cluster can be used to find fewer than \( K \) clusters. A term \( -l_1 \max(0, N_k - 1) \) is added to the \( \eta_k \) function, and \( -l_2 \max(0, M_k - 1) \) is added to the \( \mu_k \) function. This penalty can be interpreted as a significance threshold, as clusters with barely-positive scores might be found even in a dataset with no true clusters; or, it may be interpreted as a prior on the true number of clusters, which is \( \text{Geom}(\exp(-2l_1)) \) as \( K \to \infty \).
It does not change the computational complexity to use these penalties. When computing $n^k_{ij}(1)$, there is no additional computation as the penalty $l_1$ is incurred automatically. When computing $n^k_{ij}(0)$, if $y$ is the original message before accounting for $l_1$, the new message is $\max(0, y - l_1)$.

In order to set this parameter, one strategy is: randomly permute the entries of the input matrix $(\ell_{ij})$; run BCMP on the scrambled data matrix; and choose $l_1 = \frac{1}{2} \max_k \sum_{i,j} c^k_{ij} \ell_{ij}$.

### 4 The max-sum algorithm

In the following, we explain how message passing is used to optimize the BCMP objective function in approximately linear time $O(K(N + \log M)(M + \log N))$. First, we briefly explain the max-sum algorithm that we use in our optimization. Consider the following optimization:

$$\max_{X_1, \ldots, X_n} \sum_{i=1}^m f_i(x_1, \ldots, x_n)$$

where each variable $X_i$ has alphabet $\mathcal{X}_i$. We assign a function node to each function $f_i$ and a variable node to each variable. Messages $m_{X_i \rightarrow f_j}(x)$ and $m_{f_j \rightarrow X_i}(x)$ are vectors with length $|\mathcal{X}_i|$ defined as follows:

$$\begin{cases}
m_{f_j \rightarrow X_i}(x) = \max_{X_i = x} \left(f_j(X_1, \ldots, X_n) + \sum_{k \neq i} m_{X_k \rightarrow f_j}(X_j)\right), \\
m_{X_i \rightarrow f_j}(x) = \sum_{k \neq j} m_{f_k \rightarrow X_i}(x).
\end{cases}$$

When messages converge after several iterations, an optimal value of variable $X_i$ is then computed as,

$$x_i^* = \arg \max_{x \in \mathcal{X}_i} \sum_j m_{f_j \rightarrow X_i}(x).$$

If variables are binary (i.e., $|\mathcal{X}_i| = 2$), scalar messages can be passed among nodes defined as follows,

$$\begin{cases}
m_{f_j \rightarrow X_i} \triangleq m_{f_j \rightarrow X_i}(1) - m_{f_j \rightarrow X_i}(0), \\
m_{X_i \rightarrow f_j} \triangleq m_{X_i \rightarrow f_j}(1) - m_{X_i \rightarrow f_j}(0).
\end{cases}$$

When the graphical model is a tree, the max-sum algorithm reduces to the Viterbi algorithm, which is exact. On graphical models with loops, it often obtains an approximately optimal solution in practice. It is the zero-temperature version of Belief Propagation (BP), also known as the sum-product algorithm or the cavity method. BP finds marginal probabilities in a graphical model given the joint probability distribution. To find a marginal probability, all possible configurations of the other variables must be considered, by summing instead of taking a maximum over the possible configurations. The fixed points of BP correspond to local minima of the Bethe free energy [1].

The max-sum algorithm was notably used in reference [2] for a clustering algorithm, Affinity Propagation, which inspired the proposed method.

### 5 Message updates

After solving the optimization of equation (11) in the main text, there are a few details involved in computing the actual message values. Let $n$ be the value of $N_1$ that maximizes (11), and let $y$ be the solution to equation (11).
(i) For \( n_{12}^1(0) \), because the message sum is over \((i, j) \neq (1, 2)\), \(m_{12}^1 + t_{12}^1 \) is subtracted from \( s_i \). This is only significant if row 1 was included in the arg max to (11) (i.e., if \( s_1 \geq s_{(N+1-n)} \)). In that case, it might no longer be optimal to include row \( i \), in which case it may or may not be replaced by the \((n + 1)\)st largest row sum. If \( l_1 = 0 \), the message is
\[
n_{12}^1(0) = y - \min(\max(0, m_{12}^1 + t_{12}^1), s_1 - s_{(N-n)}).
\]
s_1 must be similarly adjusted for \( n_{12}^1(1) \). If \( l_1 \) is the initial penalty \( l_1 \) is used (see Supplementary Note 3.3), it might be optimal to exclude all rows and set every \( c_{12}^1 = 0 \). If \( y' \) is the message value for \( l_1 = 0 \), then accounting for \( l_1 \), the true value is \( \max(0, y' - l_1) \). For \( n_{12}^1(1) \), the arg max is unaffected, and the message value is \( y' - l_1 \).

(ii) For the message \( n_{12}^1(1) \), row \( i \) must be included whether or not it is optimal in (11), thus it is possible that one fewer row to be included. If \( s_1 < s_{(N+1-n)} \), then (i) is not relevant, and
\[
n_{12}^1(1) = y + s_1 - \min(\frac{\delta}{2}(2n + 1), s_{(N+1-n)}).
\]

(iii) For calculating \( n_{12}^1(0) \), if the initial penalty \( l_1 \) is used (see Supplementary Note 3.3), it might be optimal to exclude all rows and set every \( c_{12}^1 = 0 \). If \( y' \) is the message value for \( l_1 = 0 \), then accounting for \( l_1 \), the true value is \( \max(0, y' - l_1) \). For \( n_{12}^1(1) \), the arg max is unaffected, and the message value is \( y' - l_1 \).

### 6 Expectation-Maximization for the binary and Gaussian models

This section describes how expectation-maximization (EM) was applied to the binary and Gaussian models used in our simulations (see Section 2.4).

In the case of a binary model with parameters \( p \) and \( q \), the prior on \( p \) is \( \text{Unif}(0, 1) \). If \( n \) tuples were placed in a cluster, and \( m \) out of those \( n \) had edges, the posterior distribution for \( p \) is \( \text{Beta}(1 + m, 1 + n - m) \), and the posterior for \( q \) is computed similarly. The posteriors are independent of each other because the edge weights are assumed to be drawn independently (given the cluster assignments).

There are only two integrals to be computed:
\[
\begin{align*}
l_{ij} &= \begin{cases} 
\int_{[0,1]^2} \log \frac{p}{q} dP(p) dP(q) & e_{ij} = 1 \\
\int_{[0,1]^2} \log \frac{1-p}{1-q} dP(p) dP(q) & e_{ij} = 0
\end{cases} 
\end{align*}
\]  

(9)

Here we use \( P(p) \) and \( P(q) \) to denote the posterior CDF of these parameters. These integrals can be computed numerically by sampling from the posterior distributions.

In the case of a Gaussian model with two different means and variances, we use a normal-inverse gamma distribution on the mean and variance. Assume that \( e_{ij} = 0 \) is normally distributed with mean \( \mu_a \) and variance \( \sigma^2_a \). Let the prior distribution both within clusters and outside of them be \((\mu_a, \sigma^2_a) \sim NIG(\mu, \lambda, \alpha, \beta)\), that is,

\[
\begin{align*}
\sigma^2_a &\sim \Gamma(\alpha, \beta), \\
\mu_a | \sigma^2_a &\sim N(\mu, \sigma^2_a / \lambda).
\end{align*}
\]  

(10)

Let \( \hat{c} = (c_{ij}) \) be the current estimate of the cluster memberships. Let \( A \triangleq \{(i, j) : c_{ij} = 1\} \) and \( B \triangleq \{(i, j) : c_{ij} = 0\} \); let \( \hat{\mu}_A, \hat{\sigma}^2_A, \hat{\mu}_B, \hat{\sigma}^2_B \) be the sample means and variances of the edge weights over the estimates \( A, B \). Then the posterior distributions of the model parameters are

\[
\begin{align*}
(\mu_1, \sigma^2_1) | \hat{A} &\sim NIG\left(\frac{\hat{\mu} - 1 + |A| \hat{\mu}_A}{\lambda - 1 + |A|}, \alpha + \frac{|A|}{2}, \beta + \frac{1}{2} \hat{\sigma}^2_A\right) \quad (11) \\
(\mu_0, \sigma^2_0) | \hat{B} &\sim NIG\left(\frac{\hat{\mu} - 1 + |B| \hat{\mu}_B}{\lambda - 1 + |B|}, \alpha + \frac{|B|}{2}, \beta + \frac{1}{2} \hat{\sigma}^2_B\right)
\end{align*}
\]
Let $\phi$ be the standard Normal density function. The expression for the new input matrix is:

$$l_{ij} = \int_{R^2} \log \sigma_1^{-1} \phi(\frac{e_{ij} - \mu_1}{\sigma_1})dP(\mu_1, \sigma_1) - \int_{R^2} \log \sigma_0^{-1} \phi(\frac{e_{ij} - \mu_0}{\sigma_0})dP(\mu_0, \sigma_0)$$

(12)

This computation can be performed efficiently by drawing samples from the two distributions and reusing them for each of the $NM$ integrations. Typically, in our simulations, approximately three iterations of EM were needed for convergence.

7 Maximum Modularity

When a statistical model for the data is not available, a natural heuristic input matrix, replacing $(l_{ij})$, is the modularity matrix [3] with entries

$$M_{ij} = e_{ij} - \frac{d_i d_j}{2m},$$

where $d_i$, $d_j$ are the degrees of nodes $i$ and $j$, and $m = \frac{1}{2} \sum_{i=1}^{N} d_i$ is the total weight of the graph. The term $\frac{d_i d_j}{2m}$ is interpreted as the expected weight of $e_{ij}$ under a null model that accounts for the degree of each node; it satisfies the condition $\sum_{i=1}^{N} E(e_{ij}) = d_j$ for each column $j$ and likewise for each row $i$, where $E()$ indicates the expectation. This matrix was originally used for graph partitioning in the case of a binary, non-bipartite graph; however, it can be extended to the overlapping, weighted and bipartite cases.

In the non-partitioning case, typically most nodes will be assigned to some cluster using the maximum modularity method, for two reasons. First, if $(U_1, V_1)$ is a bicluster of modularity $x$, then there will be a “reflected” cluster $(U_2^c, V_1^c)$ that also has modularity $x$ even though it might be no denser, in terms of actual edge weights, than the network mean. Second, because the row and column sums of $M$ are all zero, there is a high probability that a node will be incorrectly assigned to any given cluster. If $V_1$ is a collection of columns and $|V_1|<<M$, then $E(\sum_{j\in V_1} M_{ij}) \approx 0$, which implies that node $i$ will be assigned to the cluster erroneously with probability approaching 1/2. To avoid finding spurious clusters and erroneously assigning extra nodes to clusters, therefore, we recommend using a cluster-size penalty that is larger than the offset. This value can be chosen by drawing a number of random “biclusters”, computing their respective modularities, and choosing a value such that all but a fraction of them have negative shifted modularity.

8 Gene expression data

We applied BCMP, ISA and LAS to three DREAM5 gene expression datasets: *In Silico*, *E. coli*, and *S. cerevisiae* [4]. We binarized these datasets, placing ones where the gene expression level was at least two standard deviations from its mean and zeroes elsewhere. We evaluated the three biclustering algorithms in terms of the total size and average density of the reported clusters; when reported clusters overlapped, these regions were not double-counted. We used BCMP to fit the stochastic block model, using different initial parameter settings for EM (see section 2.4 and supplementary section 6) and $K = 10$ (this is the number of clusters reported by LAS by default). ISA was run with different threshold-parameter settings, keeping the first ten clusters reported. LAS had no parameters to set. We found that the density of the clusters reported by LAS was diluted by a few large, low-density clusters. ISA and BCMP had similar results for the *In Silico* and *E. coli* datasets, and BCMP reported denser clusters on the *S. cerevisiae* datasets (see Supplementary Figure 2).

9 Document classification example

Biclustering has been used for document clustering [5]. Here we illustrate of how BCMP might be applied to such a problem.

Let $D = d_1, ..., d_N$ be a collection of documents containing the words $W = w_1, ..., w_M$, and let $(D_1, W_1), ..., (D_K, W_K)$ be a hidden set of topics (each word and document can belong to any
Figure 2: Total size and average density of biclusters reported by BCMP, ISA and LAS for three gene expression datasets. a) *In Silico*. b) *E. coli*. c) *S. cerevisiae*.

number of topics). Document $d_i$ contains $n_{ij}$ instances of word $w_j$ out of $n_i = \sum_j n_{ij}$ total words. Assume that $n_i$ is large enough to use a Poisson model: $n_{ij} \sim \text{Poisson}(\lambda_{ij})$ where $\lambda_{ij} = n_i r_j$ if document $d_i$ concerns a topic relating to word $w_j$ and $\lambda_{ij} = n_i s_j$ otherwise. Assume that $K$ is given.

The problem is to recover the topics. It can be solved by estimating the parameters $r_j, s_j$ and biclustering the bipartite graph $(D, W, (n_{ij}))$.

We can obtain initial estimates for $r_j$ and $s_j$ using k-means on $n_{1j}, ..., n_{Nj}$. This is effectively a clustering problem along one axis with two clusters:

$$\max_{r_j, s_j, c_{1j}, ..., c_{Nj}} P(n_{1j}, ..., n_{Nj} | r_j, s_j, c_{1j}, ..., c_{Nj})$$

where $c_{ij}$ is the indicator variable for document $i$ concerning some topic relating to word $j$. Words with no evidence for $r_j \neq s_j$ at some significance threshold can be discarded. After performing biclustering, these estimates can be improved using the newly estimated cluster memberships.

After estimating $r_j$ and $s_j$, the likelihood ratios are

$$\frac{p_{ij}}{q_{ij}} = \frac{P(n_{ij} | \lambda_{ij} = n_i r_j)}{P(n_{ij} | \lambda_{ij} = n_i s_j)} = \frac{(n_i r_j)^{n_{ij}} e^{-n_i r_j} / n_{ij}!}{(n_i s_j)^{n_{ij}} e^{-n_i s_j} / n_{ij}!}$$

and the input matrix for BCMP is

$$\ell_{ij} = \max(0, \log \frac{p_{ij}}{q_{ij}} + \delta) = \max(0, n_{ij} \log(\frac{r_j}{s_j}) + n_i(s_j - r_j) + \delta). \quad (13)$$

EM can be used to iteratively update the input matrix and improve the parameter estimates as discussed in Section 2.4.
Figure 3: Running time for BCMP. In Proposition 3, we showed that the complexity of a round of message updates for BCMP is nearly linear, i.e., $O(K(N + \log M)(M + \log N))$. This figure illustrates the running time of BCMP on a personal computer, with $N = M$.

10 Pseudocodes for BCMP

The pseudocode is presented as three functions. The first calls the two message-update functions, computes clusters from messages, and checks for convergence.

The second function computes messages from the $\mu_k$ and $\eta_k$ function nodes. The steps for computing these messages are explained here briefly; see Section 2.3 and Supplementary Note 5 for a derivation of the message update rules. An optimization is performed under two constraints: either tuple $(i, j)$ must be assigned to cluster $k (c_{i,j}^k = 1)$, or it must not be. While the optimization is over $NM$ variables, only the number of rows or columns (for $\eta$ and $\mu$ respectively) is penalized; thus, the optimum is achieved by including every tuple $(i, j)$ with positive outgoing message for some number of rows $i$ (or columns $j$). When tuple $(i, j)$ is constrained to be in the cluster, node $i$ must also be in the cluster; however, node $i$ may still be in the cluster even when tuple $(i, j)$ is constrained not to be. The steps are, first, to compute the sum of the positive incoming messages for each node $i = 1, \ldots, N$ and sort them. Second, to find the unconstrained $\arg\max$ by comparing the sorted list of message sums with the marginal penalty for including another node in the cluster. Three indices are computed: $t_0, t_1, t_2$, which are the numbers of nodes included in the cluster under various constraints. $t_1$ is the index for the unconstrained maximum. When an extra node, not included in the unconstrained maximum, is constrained to be included, $t_2 \leq t_1$ is the optimal number of additional nodes to also include. $t_0 \in \{t_1, t_1 + 1\}$ is the optimal number when a node drops out of the sum (owing to one of the tuples being excluded). The third step is to compute the constrained $\arg\max$ for each tuple; one of the constraints gives the unconstrained maximum, and the other gives an optimization that is solved by including either $t_0$ or $t_2$ nodes from the sorted list of message sums.

The third function computes messages from the $\tau_{ij}$ function nodes. First, it finds the largest and second largest messages $msg_k$, and the $\arg\max k_1$; the second largest message value is needed to compute the message for $k = k_1$. Then, it computes the message values explicitly.
Algorithm 1 Biclustering Using Message Passing

function BCMP((l_{ij}), K, l_0, r)

Require: N × M dataset (l_{ij}); maximum number of clusters to find K; penalty per nonempty cluster l_1.

(t^1_{ij}) ← 0
(n^k_{ij}) ← random
(m^k_{ij}) ← 0

▷ small random values

for rep = 1, ..., repmax do
    for i = 1, ..., N do
        for j = 1, ..., M do
            (t^1_{ij},...,t^k_{ij}) ← (t^1_{ij},...,t^k_{ij}) · \lambda_1 + tuple_update(n^1_{ij} + m^1_{ij},...,n^k_{ij} + m^k_{ij}) · (1 - \lambda_1)
        end for
    end for
    for k = 1, ..., K do
        (n^1_{ij},...,n^k_{ij}) ← (n^1_{ij},...,n^k_{ij}) · \lambda + pen_update((t^k_{ij}) + (n^k_{ij}), l_0 r_k) · (1 - \lambda)  
    end for
    ▷ Compute biclusters from current messages in order to update parameters and check for convergence.

(c^k_{ij}) ← (n^k_{ij} + m^k_{ij} + t^k_{ij} > 0)  
(ind_{ij}) ← (\sum_k c^k_{ij} > 0)  

▷ N × M × K boolean array  
▷ N × M boolean array

score ← \sum_{ij} ind_{ij} (l_{ij} - 2l_0)

if score > oldscore then
    oldscore ← score
else
    break
end if
end for

return (t^k_{ij}) + (n^k_{ij}) + (m^k_{ij})

end function

\(\triangle\lambda\) is a damping factor; a good value is \(\lambda = 1/2\). The input to the penalty message update function is a N×M or M×N matrix, and the penalty coefficient.
Algorithm 2 Message update function for $\eta$ and $\mu$

function PEN\_UPDATE\((msg_{ij})\), $a$, $b$

Require: $(msg_{ij})$ the $N \times M$ matrix of messages from the variable nodes; $a$ the penalty coefficient; $b$ the initial penalty per nonempty cluster.

$(sums_1, ..., sums_N) \leftarrow (\sum_j \max(0, msg_{1i}), ..., \sum_j \max(0, msg_{Ni}))$ \hfill\(^\triangleright\) Step one

sort ($sums$) and store the permutation as $\sigma : 1, ..., N \rightarrow 1, ..., N$

$(t_1, ..., T_N) \leftarrow a \cdot (1, 3, 5, ..., 2N - 1)$ \hfill\(^\triangleright\) $T_i = ai^2 - a(i - 1)^2$ the marginal penalty \hfill\(^\triangleright\) Step two

$t_1 \leftarrow \min\{\{n : msg_{\sigma(n)} - T_n < 0\}\} - 1$ \hfill\(^\triangleright\) or $N$ if this set is empty

$t_2 \leftarrow \min\{\{n : sums_{\sigma(n)} - T_{n+1} < 0\}\} - 1$ \hfill\(^\triangleright\) or $N$ if this set is empty

if $0 < t_1 < N$ then

if $sums_{\sigma(n)} \geq T_{t_1}$ then

$t_0 \leftarrow t_1 + 1$

else

$t_0 \leftarrow t_1$

end if

else if $t_1 = 0$ then

for $i = 1, ..., N$ do

$newmsg_{1i}, ..., newmsg_{Mi} \leftarrow b - a + sums_i - (\max(0, msg_{1i}), ..., \max(0, msg_{Ni}))$

end for

return $(newmsg_{ij})$

else

$t_0 \leftarrow t_1$

end if

$sum\_to\_t_0 \leftarrow \sum_{i=1}^{t_0} sums_{\sigma(i)}$

$sum\_to\_t_1 \leftarrow \sum_{i=1}^{t_1} sums_{\sigma(i)}$

$sum\_to\_t_2 \leftarrow \sum_{i=1}^{t_2} sums_{\sigma(i)}$

$unconstrained\_max \leftarrow b + sum\_to\_t_1 - a \cdot t_1^2$ \hfill\(^\triangleright\) Step three

for $i = 1, ..., t_1$ do

for $j = 1, ..., M$ do

$newmsg_{\sigma(i),j} \leftarrow unconstrained\_max - \max(0, msg_{\sigma(i),j}) - \max(b + max(0, sum\_to\_t_1 - \max(0, msg(\sigma(i), j)) - a \cdot t_1^2, sum\_to\_t_0 - sums_{\sigma(i)} - a \cdot (t_0 - 1)^2))$

end for

end for

unconstrained\_max $\leftarrow \max(0, unconstrained\_max)$

for $i = t_1 + 1, ..., N$ do

for $j = 1, ..., M$ do

$newmsg_{\sigma(i),j} \leftarrow b + sum\_to\_t_2 + sums_{\sigma(i)} - \max(0, msg_{\sigma(i),j}) - a \cdot (t_2 + 1)^2 - unconstrained\_max$

end for

end for

return $(newmsg)$

end function
Algorithm 3  Message update function for $\tau$

```java
function TUPLE_UPDATE((msgk), δ, ℓij)
    for $k = 1, ..., K$ do
        if $w1 <= msgk$ then
            $k1 \leftarrow k$
            $w2 \leftarrow w1$
            $w1 \leftarrow msgk$
        else if $w2 <= msgk$ then
            $w2 \leftarrow msgk$
        end if
    end for
    for $k = 1, ..., K$ do
        if $k \neq k1$ then
            $newmsgk = \ell_{ij} - \max(0, \delta + msgk) - \left[\max(0, \ell_{ij}) + w1 - \max(0, \delta + msgk) - \max(0, 2\delta + w1)\right]$
        else
            $newmsgk = \ell_{ij} - \max(0, \delta + msgk) - \left[\max(0, \ell_{ij}) + w2 - \max(0, \delta + msgk) - \max(0, \delta + w2)\right]$
        end if
    end for
    return (newmsg)
end function
```

References


