Finding biclusters by random projections

Stefano Lonardi[1], Wojciech Szpankowski[2], and Qiaofeng Yang[1]

[1] Dept. of Computer Science – University of California – Riverside, CA
[2] Dept. of Computer Sciences – Purdue University – West Lafayette, IN

Abstract. Given a matrix X composed of symbols, a bicluster is a submatrix of X obtained by removing some of the rows and some of the columns of X in such a way that each row of what is left reads the same string. In this paper, we are concerned with the problem of finding the bicluster with the largest area in a large matrix X. The problem is first proved to be **NP**-complete. We present a fast and efficient randomized algorithm that discovers the largest bicluster by random projections. A detailed probabilistic analysis of the algorithm and an asymptotic study of the statistical significance of the solutions are given. We report results of extensive simulations on synthetic data.

1 Introduction

Clustering refers to the problem of finding a partition of a set of input vectors, such that the vectors in each subset (cluster) are "close" to one another (according to some predefined distance). A common limitation to the large majority of clustering algorithms is their inability to perform on high dimensional spaces (see, e.g., [1, 2]).

Recent research has focused on the problem of finding hidden sub-structures in large matrices composed by thousands of high dimensional vectors (see, e.g., [3–10]). This problem is known as *biclustering*. In biclustering, one is interested in determining the similarity in a *subset* of the dimensions (subset that has to be determined as well). Although there exists several definitions of biclustering, it can be informally described as the problem of finding a partition of the vectors and a subset of the dimensions such that the projections along those directions of the vectors in each cluster are close to one another. The problem requires to cluster the vectors and the dimensions simultaneously, thus the name "biclustering".

Biclustering has important applications in several areas, such as data mining, machine learning, computational biology, and pattern recognition. Data arising from text analysis, market-basket data analysis, web logs, etc., is usually arranged in a contingency table or co-occurrence table, such as, a word-document table, a product-user table, a cpu-job table or a webpage-user table. Discovering a large bicluster in a product-user matrix indicates, for example, which users share the same preferences. Finding biclusters has therefore applications in recommender systems and collaborative filtering, identifying web communities, load balancing, discovering association rules, among others.

In computational biology, this problem is associated with the analysis of gene expression data obtained from microarray experiments. Gene expression data is typically arranged in a table with rows corresponding to genes, and columns corresponding to patients, tissues, time points, etc. The classical approach to analyze microarray data is clustering. The process of clustering partitions genes into mutually exclusive clusters under the assumption that genes that are involved in the same genetic pathway behave similarly across all the testing conditions. The assumption might be true when the testing conditions are associated with time points. However, when the testing conditions are heterogeneous, such as patients or tissues, the previous assumption is not appropriate anymore. One would expect that a group of genes would exhibit similar expression patterns only in a subset of conditions, such as the subset of patients suffering from the same type of disease. Under this circumstance, biclustering becomes the alternative to the traditional clustering paradigm. The results of biclustering may enable one to discover hidden structures in gene expression data in which many genetic pathways might be embedded. It might also allow one to uncover unknown genetic pathways, or to assign functions to unknown genes in already known genetic pathways.

Biclustering is indeed, not a new problem. In fact, it is also known under several other names, namely "co-clustering", "two-way clustering" and "direct clustering". The problem was first introduced in the seventies in a paper by Hartigan [11]. Almost thirty years later, Cheng and Church [3] raised the interest on this problem for applications in gene expression data analysis.

Several other researchers studied the problem recently. Wang *et al.* propose the *pCluster* model that is capable of discovering shifting or scaling patterns from raw data sets [4]. Tanay *et al.* [5] combine a graph-theoretic approach with a statistical modeling of the data to discover biclusters in large gene expression datasets. Ben-Dor *et al.* [6] introduce a new notion of a bicluster called *order preserving submatrix*, which is a group of genes whose expression level induces a linear ordering across a subset of the conditions. Murali and Kasif [12] (see also [10]) propose the concept of *xmotif*, which is defined as a subset of genes whose expression is simultaneously conserved for a subset of samples.

As we were writing this document, we became aware of two other contributions to the subject, by Sheng *et al.* [8], and Mishra *et al.* [9], that use a randomized approach similar with the work described here. Sheng *et al.* [8] propose a randomized algorithm based on Gibbs sampling to discover large biclusters in gene expression data. Their model of a bicluster is probabilistic, that is, each entry of the matrix is associated with a probability. Mishra *et al.* [9] are concerned with the problem of finding ϵ-bicliques which maximizes the number of edges[3]. Given a bipartite graph (U, V, E), a subgraph (U', V') is ϵ-biclique if each vertex in U' is a neighbor of at least $(1 - \epsilon)$ fraction of vertices in V'. The authors give an efficient randomized algorithm that finds the largest ϵ-biclique, but no experimental results are reported.

[3] the connection between bicliques and bicluster will be explained in detail in Section 2

As shown in papers [12] and [8], the problem of biclustering gene expression data can be formulated on a discrete domain, by first discretizing the gene expression matrix into a matrix over a finite alphabet. The simplifying assumption is that the set of states in which each gene operates is finite, such as up-regulated, down-regulated or unchanged. Once the data is discretized into strings where each symbol corresponds to a state, the biclustering problem reduces to the problem of finding a subset of the rows and a subset of the columns such that the submatrix induced has the property that each row reads the same string. Such a submatrix would therefore correspond to a group of genes that exhibit a coherent pattern of states over a subset of conditions. This is indeed the formulation of the problem that we define in Section 2, which is first proved to be **NP**-complete. In Section 3 we present a randomized algorithm which is efficient and easy to understand and implement. Section 4 presents an asymptotic analysis that allows one to determine the statistical significance of the solution. Finally, in Section 5 we report simulation results on synthetic data.

2 Notations and problem definition

We use standard concepts and notation about strings. The set Σ denotes a nonempty *alphabet* of *symbols* and a *string* over Σ is an ordered sequence of symbols from the alphabet. We use the variable a as a shorthand for the cardinality of the set Σ, that is, $a = |\Sigma|$. Given a string x, the number of symbols in x defines the *length* $|x|$ of x.

Similarly, we can define a two-dimensional $n \times m$ string (or matrix) $X \in \Sigma^{n \times m}$ over the alphabet Σ. The element (i, j) of X is denoted by $X_{[i,j]}$. A *row selection* of size k of X is defined as the subset of the rows $R = \{i_1, i_2, \ldots, i_k\}$, where $1 \leq i_s \leq n$ for all $1 \leq s \leq k$. Similarly, a *column selection* of size l of X is defined as a subset of the columns $C = \{j_1, j_2, \ldots, j_l\}$, where $1 \leq j_t \leq m$ for all $1 \leq t \leq l$.

The submatrix $X_{(R,C)}$ *induced* by the pair (R, C) is defined as the matrix

$$X_{(R,C)} = \begin{vmatrix} X_{[i_1,j_1]} & X_{[i_1,j_2]} & \cdots & X_{[i_1,j_l]} \\ X_{[i_2,j_1]} & X_{[i_2,j_2]} & \cdots & X_{[i_2,j_l]} \\ \cdots & \cdots & \cdots & \cdots \\ X_{[i_k,j_1]} & X_{[i_k,j_2]} & \cdots & X_{[i_k,j_l]} \end{vmatrix}$$

Given a selection of rows R, we say that a column j, $1 \leq j \leq m$, is *clean* with respect to R if the symbols in the j-th column of X restricted to the rows R, are identical.

The problem addressed in this paper is defined as follows.

LARGEST BICLUSTER(f) problem
Instance: A matrix $X \in \Sigma^{n \times m}$ over the alphabet Σ.
Question: Find a row selection R and a column selection C such that the rows of $X_{(R,C)}$ are identical strings and the objective function $f(X_{(R,C)})$ is maximized.

Some examples of objective functions are the following.

- $f_1\left(X_{(R,C)}\right) = |R| + |C|$
- $f_2\left(X_{(R,C)}\right) = |R|$ provided that $|C| = |R|$
- $f_3\left(X_{(R,C)}\right) = |R|\,|C|$

The problem in general may have multiple solutions which optimize the objective function. The solutions may also "overlap", that is, they may share some elements of the original matrix.

The computational complexity of this family of problems depends on the objective function f. In the literature, the problem has been studied mostly from a graph-theoretical viewpoint which corresponds to the special case $\Sigma = \{0, 1\}$. In fact, observe that a matrix $X \in \{0,1\}^{n \times m}$ is the adjacency matrix of a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = n$ and $|V_2| = m$. An edge $(i, j) \in E$ connects node $i \in V_1$ to node $j \in V_2$ if $X_{i,j} = 1$. Thus, a submatrix of 1's in X corresponds to a subgraph of G which is completely connected. Such a subgraph is called a *biclique*. Because of this relation, we use the terms "submatrix", "biclique", and "bicluster" interchangeably.

When the alphabet is binary and we are looking for the largest submatrix composed only by 1's [4], the LARGEST BICLUSTER reduces to well-known problems on bipartite graphs. More specifically, the LARGEST BICLUSTER problem associated with objective function f_1 is known as the MAXIMUM VERTEX BICLIQUE problem, and it can be solved in polynomial time because it is equivalent to the maximum independent set in bipartite graphs which, in turn, can be solved by a minimum cut algorithm (see, e.g., [13]). The same problem with objective function f_2 over a binary alphabet is called BALANCED COMPLETE BIPARTITE SUBGRAPH problem or BALANCED BICLIQUE problem and it is listed as GT24 among the **NP**-complete problems in Garey & Johnson's book [14] (see also [15]).

The LARGEST BICLUSTER problem with objective function f_3 and $\Sigma = \{0, 1\}$ is called MAXIMUM EDGE BICLIQUE problem. The problem requires to find the biclique which has the maximum number of edges. The problem is proved to be **NP**-complete in [16] by reduction from 3SAT. The weighted version of this problem is shown **NP**-complete by Dawande *et al.* [17].

In [13] Hochbaum studies a problem related to MAXIMUM EDGE BICLIQUE, which is the problem of finding the number of edges that need to be deleted so that the resulting graph is a biclique. Hochbaum describes a 2-approximation algorithm based on LP-relaxation. According to Pasechnik [18] this approximation ratio does not hold for the original MAXIMUM EDGE BICLIQUE problem. Pasechnik shows a semidefinite relaxation, and claims that his relaxation is in general better than [13].

The following theorem establishes the hardness of the problem of finding the largest area bicluster over a general alphabet. For lack of space the proof is omitted.

[4] In general, a solution of the largest bicluster can contain a column of zeros, as long as they appear in all rows of the submatrix

Theorem 1. *The decision problem associated with* LARGEST BICLUSTER(f_3) *is* ***NP****-complete.*

By the same approach, LARGEST BICLUSTER(f_2) can also be proved to be **NP**-complete. In the rest of this paper we will concentrate our attention on the problem of finding the largest-area bicluster. For practical reasons that will become apparent in Section 3, the objective function that we are maximizing is

$$\tilde{f}_3\left(X_{(R,C)}, \hat{r}, \hat{c}\right) = |R|\,|C| \text{ provided that } |R| \geq \hat{r} \text{ and } |C| \geq \hat{c}$$

where $\hat{r}$ and $\hat{c}$ are two input parameters.

3 Randomized search

Given that LARGEST BICLUSTER(f_3) problem is **NP**-complete, it is unlikely that a polynomial time algorithm could be found. In this paper, we present a randomized algorithm which finds a maximal solution with probability $1 - \epsilon$, where $0 < \epsilon < 1$.

Assume that we are given a large matrix $X \in \Sigma^{n \times m}$ in which a submatrix $X_{(R^*,C^*)}$ is implanted. Assume also that the submatrix $X_{(R^*,C^*)}$ is maximal. To simplify the notation, let $r^* \equiv |R^*|$ and $c^* \equiv |C^*|$.

The idea behind the algorithm comes from the following simple observation. Observe that if we knew R^*, then C^* could be determined by selecting the clean columns with respect to R^*. If instead we knew C^*, then R^* could be obtained by taking the maximal set of rows which read the same string. Unfortunately, neither R^* nor C^* is known. Our approach is to "sample" the matrix by random projections, with the expectation that at least some of the projections will overlap with the solution (R^*, C^*). Clearly, one can project either rows or columns. In what follows we describe how to retrieve the solution by sampling columns.

The algorithm works as follows. Select a random subset S of size k uniformly from the set of columns $\{1, 2, \ldots, m\}$. Assume for the time being that $S \cap C^* \neq \emptyset$. If we knew $S \cap C^*$, then (R^*, C^*) could be determined by the following three steps (1) select the string(s) w that appear exactly r^* times in the rows of $X_{[1:n, S \cap C^*]}$, (2) set R^* to be the set of rows in which w appears and (3) set C^* to be the set of clean columns corresponding to R^*.

The algorithm would work, but there are a few problems that are still unresolved. First, the set $S \cap C^*$ could be empty. The solution is to try different random projections S, relying on the argument that the probability that $S \cap C^* \neq \emptyset$ *at least once* will approach one with more and more projections. The second problem is that we do not really know $S \cap C^*$. But, certainly $S \cap C^* \subseteq S$, so our approach is to check all possible subsets $U \subseteq S$ such that $|U| \geq k_{\min}$, where $1 \leq k_{\min} \leq k$ is a user-defined parameter. The final problem is that we assumed that we knew r^*, but we do not. The solution is to introduce a *row threshold* parameter, called $\hat{r}$, that replaces r^*.

As it turns out, we need another parameter to avoid producing solutions with too few columns. The *column threshold* $\hat{c}$ is used to discard submatrices whose

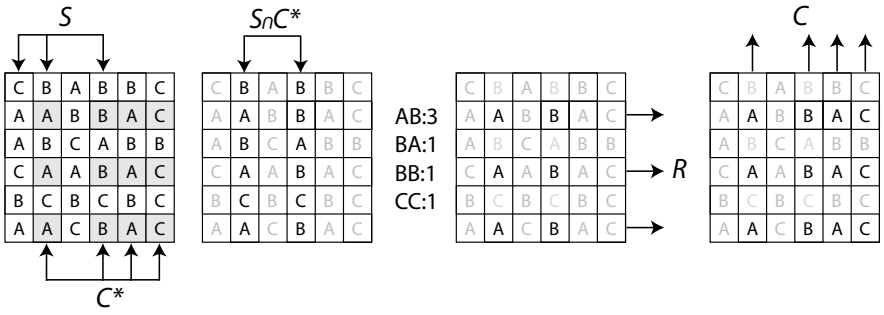

Fig. 1. An illustration of a recovery of the embedded matrix by random projections. C^* is the set of columns containing the embedded submatrix. S is a random selection of columns. By following the steps described in the text, the correct solution can be easily retrieved.

number of columns is smaller than $\hat{c}$. The algorithm considers all the submatrices which satisfy the user-defined row and column threshold as candidates. Among all candidate submatrices, only the ones that maximize the total area are kept. A sketch of the algorithm is shown in Figure 2. As noted in the introduction, a very similar strategy was developed independently and concurrently by Mishra *et al.* [9].

The algorithm depends on five key parameters, namely the projection size k, the minimum subset size $k_{\min}$, the row threshold $\hat{r}$, the column threshold $\hat{c}$, and the number of iterations t. We discuss how to choose each of these in the rest of the section.

Parameter Selection. The projection size k is determined by a probabilistic argument. It is well-known that in a random string of size m over an alphabet of size a, the number of occurrences of substrings has two different probabilistic regimes (1) Gaussian distributed for strings shorter than $\log_a m$ and (2) Poisson distributed for strings longer than $\log_a m$ (see, e.g., [19]). Based on this observation, when $k_{\min} = k$ we argue that $k = \log_a m$ is the optimal trade-off between generating too many trivial solutions (k too small) and potentially missing the solution (k too large). This value of k has been confirmed to be the optimal choice in our simulations. When $k_{\min} = 1$, then k can be chosen significantly larger, but this will adversely affect the running time. An experimental comparison between $k_{\min} = k$ (i.e., no subsets), and $k_{\min} = 1$ (i.e., all subsets) is reported in Section 5.1.

The thresholds $\hat{r}$ and $\hat{c}$ are associated with the uncertainty on the size of the largest submatrix r^*, c^* for a particular input instance. There may be situations in which the user has already a good idea about r^*, c^*. If however r^* and c^* are completely unknown, then our target will be to find "statistically significant" biclusters. In Section 4 we will present a theorem (Thorem 2) which gives the expected number of columns of the largest submatrix in a random matrix, when

```
LARGEST_BICLUSTER_C(X, t, k, k_min, r̂, ĉ)
INPUT: X is a n × m matrix over Σ
       t is the number of iterations
       k is the projection size
       k_min is the size of the smallest subset of the projection
       r̂, ĉ are the "thresholds" on the number of rows and columns, resp.
1   repeat t times
2       select randomly a subset S of columns such that |S| = k
3       for all subsets U ⊆ S such that |U| ≥ k_min do
4           D ← all strings induced by X_[1:n,U] that appear at least r̂ times
5           for each string w in D
6               V ← rows corresponding to w
7               Z ← all "clean" columns corresponding to V
8               if |Z| ≥ ĉ then save (V, Z)
9   return the (V, Z) that maximizes f
```

Fig. 2. A sketch of the algorithm that discovers large biclusters (sampling columns)

the number of rows is fixed. Based on this, we propose the following trial-and-error strategy. Set $\hat{r}$ to some value between 1 and n, and use Theorem 2 to set the value $\hat{c}$. Run the algorithm. If the algorithm returns too many solutions, try to increase $\hat{r}$ and update $\hat{c}$ correspondingly. If there are no solutions, lower the value of $\hat{r}$ and repeat. Observe that the number of choices for $\hat{r}$ is finite since $\hat{r} \in [1, n]$. By using Theorem 2 to set the threshold $\hat{c}$, we are trying to filter out submatrices whose size is small enough that they could appear in totally random matrices.

Because of the randomized nature of the approach, there is no guarantee that the algorithm will find the solution after a given number of iterations. We therefore need to choose t so that the probability that the algorithm will recover the solution in at least one of the t trials is $1-\epsilon$, where $0 < \epsilon < 1$ is a user-defined parameter.

Let $\alpha(n, m, k, r^*, c^*, a)$ be the probability of missing the solution in one of the trials assuming that r^* and c^* are known and that $k_{\min} = 1$. There are two disjoint cases in which the algorithm can miss (R^*, C^*). The first is when the random projection S misses completely C^*, i.e., $S \cap C^* = \emptyset$. The second is when $S \cap C^* = U \neq \emptyset$ but the string w chosen by the algorithm among the rows $X_{[1:n,U]}$ also appears in another row that does not belong to the set R^* of the real solution. In this case, the algorithm will select a set of rows larger than R^* and thus miss the solution. Hence, we have

$$\alpha(n, m, k, r^*, c^*, a) = \mathsf{Pr}\{S \cap C^* = \emptyset\} + \sum_{i=1}^{k} \mathsf{Pr}\{|S \cap C^*| = i \text{ and } |R| > r^*\}$$

ϵ	$a=2,k=8$	$a=4,k=4$	$a=8,k=3$	$a=16,k=2$	$a=32,k=2$
0.005	18794	1342	306	179	99
0.05	10626	759	173	101	56
0.1	8168	583	133	78	43
0.2	5709	408	93	54	30
0.3	4271	305	70	41	23
0.4	3250	232	53	31	17
0.5	2459	176	40	23	13
0.6	1812	129	29	17	10
0.7	1265	90	21	12	7
0.8	792	57	13	8	4
0.9	374	27	6	4	2

Table 1. The estimated number of iterations for a matrix 256×256 with a submatrix 64×64, for different choices of ϵ, alphabet size a, and projection size k (sampling columns)

$$= \mathsf{Pr}\{S \cap C^* = \emptyset\} + \sum_{i=1}^{k} \mathsf{Pr}\{|R| > r^* \text{ given } |S \cap C^*| = i\}\mathsf{Pr}\{|S \cap C^*| = i\}$$

Let Y be the random variable associated with the size of the set $S \cap C^*$, that is, $Y = |S \cap C^*|$. Since we are sampling without replacement, Y follows the hyper-geometric distribution.

$$\mathsf{Pr}\{Y = 0\} = \binom{m-c^*}{k} \Big/ \binom{m}{k} \quad \text{and} \quad \mathsf{Pr}\{Y = i\} = \binom{c^*}{i}\binom{m-c^*}{k-i} \Big/ \binom{m}{k}$$

In order to compute the probability of missing the solution given $|S \cap C^*| = i$, we have to estimate how likely a string w belonging to some of the rows of $X_{[1:n,U]}$ is more frequent than r^*. Assuming the symbols in the matrix X are generated by a symmetric Bernoulli i.i.d. model, the probability that w will never appear in the other $n - r^*$ rows is $\left(1 - \frac{1}{a^i}\right)^{n-r^*}$ and therefore

$$\mathsf{Pr}\{|R| > r^* \text{ given } |S \cap C^*| = i\} = 1 - \left(1 - \frac{1}{a^i}\right)^{n-r^*}$$

Combining all together, the probability of missing the solution in one iteration is given by

$$\alpha(n, m, k, r^*, c^*, a) = \frac{\binom{m-c^*}{k} + \sum_{i=1}^{k} \left(1 - \left(1 - \frac{1}{a^i}\right)^{n-r^*}\right)\binom{c^*}{i}\binom{m-c^*}{k-i}}{\binom{m}{k}}$$

Now suppose we want the probability of missing the solution to be smaller than a given ϵ, $0 < \epsilon < 1$. We can obtain the number of iterations t by solving the inequality $(\alpha(n, m, k, r^*, c^*, a))^t \leq \epsilon$, which gives

$$t \geq \frac{\log \epsilon}{\log \alpha(n, m, k, r^*, c^*, a)} \tag{1}$$

This bound on the number of iterations has been verified by our experimental results (compare Table 1 with our experimental results shown in Figure 3). For example, by setting $a = 4$, $k = 4$, $\epsilon = 0.7$, equation (1) gives $t = 90$ iterations whereas the experimental results show that with 90 iterations we obtain a performance of $\epsilon = 0.689$.

The worst case time complexity of LARGEST_BICLUSTER_C is bounded by $O\left(t \sum_{j=k_{\min}}^{k} \binom{k}{j} (kn + nm)\right)$. If $k_{\min} = 1$, then the time complexity becomes $O\left(t2^k(kn + nm)\right)$. Although the complexity is exponential in k, choosing k to be $O(\log_a m)$ makes the algorithm run in $O\left(tm^{1/\log_2 a}(kn + nm)\right)$ time.

The probability of missing the solution changes significantly when we set $k_{\min} = k$. In this case, we are not checking any of the subsets of S, but we simply rely on the fact that eventually one of the random projections S will end up completely contained in C^*, in which case we have a chance to find the solution.

Since we avoid checking the $O(2^k)$ subsets of S, the number of iterations t to achieve the same level of performance of the case $k_{\min} = 1$ must be significantly larger. Indeed, by a similar argument as we did for $k_{\min} = 1$, the probability of missing the solution when $k_{\min} = k$ can be estimated by the following formula

$$\begin{aligned}
\tilde{\alpha}(n, m, k, r^*, c^*, a) &= \mathsf{Pr}\{|S \cap C^*| < k\} + \mathsf{Pr}\{|S \cap C^*| = k \text{ and } |R| > r^*\} \\
&= 1 - \mathsf{Pr}\{|S \cap C^*| = k\} + \mathsf{Pr}\{|S \cap C^*| = k \text{ and } |R| > r^*\} \\
&= 1 - \left(\binom{c^*}{k} / \binom{m}{k}\right) + \left(\left(1 - \left(1 - \frac{1}{a^k}\right)^{n-r^*}\right) \binom{c^*}{k} / \binom{m}{k}\right) \\
&= 1 - \left(\left(1 - \frac{1}{a^k}\right)^{n-r^*} \binom{c^*}{k} / \binom{m}{k}\right)
\end{aligned}$$

As mentioned above, we also have the option to project the rows instead of the columns, which would result in a slightly different algorithm that we called LARGEST_BICLUSTER_R. The details and the analysis of this algorithm will be reported in the journal version of this manuscript.

Both strategies were implemented and tested extensively. Results are reported in Section 5.

4 Statistical analysis

We now analyze the statistical significance of finding a large submatrix of size $r \times c$ hidden into a random $n \times m$ matrix over an alphabet of cardinality a. More specifically, we randomly generate a matrix $X \in \Sigma^{n \times m}$ using a memoryless source with parameters $\{p_1, \ldots, p_a\}$ where p_i is the probability of the i-th symbol in Σ. Given X, the goal is to characterize asymptotically the size of the largest submatrix in X.

rows	columns observed	columns predicted
1	256	256
2	160	165.6771209
3	100	103.9626215
4	67	67.24371945
5	45	44.84053788
6	31	30.70906224
7	23	21.48364693
8	16	15.26873716

Table 2. The statistics of large submatrices in a random $\{0,1\}$-matrix of size 256×256. The second column reports the number of columns of the submatrices observed in a random matrix, whereas the third reports the prediction based on Theorem 2

For convenience of notation, let us call $P_r = p_1^r + p_2^r + \ldots + p_a^r$ the probability of observing a clean column over r rows, and let us define $H(x) = -x \ln x - (1-x)\ln(1-x)$.

The first result characterizes the random variable associated with the number of columns of the largest bicluster, when we fix the number of rows. Both proofs are omitted due to lack of space.

Theorem 2. *Let $\boldsymbol{C}_{n,m,r,a}$ be the random variable associated with the number of columns of the submatrix with the largest area in a matrix $X \in \Sigma^{n \times m}$ generated from a memoryless source, once the number of rows r is fixed. Then*

$$\boldsymbol{C}_{n,m,r,a} \leq mP_r + \sqrt{2P_r(1-P_r)mF(n,r)} \equiv C_{\max}$$

with high probability and as $n \to \infty$, where

$$F(n,r) = \begin{cases} r \log n & \text{if } r = o(n) \\ nH(\alpha) & \text{if } r = \alpha n \text{ where } 0 < \alpha < 1 \end{cases}$$

When $r = o(n)$ the error term is $O(1/\log^d n)$ for some $d > 1$ that may depend on a, whereas the error becomes $O(1/\sqrt{n})$ when $r = \alpha n$. The prediction on random matrices is indeed quite accurate as reported in Table 2. We claim that the upper bound is actually an equality, that is, asymptotically and with high probability $\mathbf{C}_{n,m,r,a} = C_{\max}$.

The practical implications of Theorem 2 are twofold. First, the expected number of columns can be used to set the column threshold parameter $\hat{c} \gg \max\{C_{\max}, 1\}$. That allows the algorithm to avoid considering statistically non-significant submatrices. Second, observe that when $\log n = o(m)$, then the dominant term of $C_{\max}$ is the average, say $\mathbf{E}[C]$, of the number of clean columns, that is, $\mathbf{E}[C] = mP_r$. This implies $C_{\max}/\mathbf{E}[C] \leq 1 + o(1)$ for $\log n = o(m)$, and therefore with high probability any algorithm is asymptotically optimal. Clearly, this is not true for $r = \alpha n$. Finally, in passing we add that when we restrict the

search to largest *squared* matrix (see objective function f_2 above), then its side is asymptotically equal to $2\log(n/(2\log n))/\log P_r^{-1}$.

The second result characterizes the random variable associated with the area of the solution. For convenience of notation, given a memoryless source with parameters $\{p_1, \ldots, p_a\}$ we define $p_{\max} = max_{1 \le i \le a} p_i$.

Theorem 3. *Let $\boldsymbol{A}_{n,m,a}$ be the random variable associated with the area of the largest submatrix in a matrix $X \in \Sigma^{n \times m}$, $m \le n$, generated from a memoryless source. Then, with high probability for any $\epsilon > 0$ and as $n \to \infty$*

$$\boldsymbol{A}_{n,m,a} \le (1+\epsilon) rc$$

where $r = n/2$ and $c = 2\ln 2/\ln p_{\max}^{-1}$.

The intuition behind Theorem 3 is that on random matrices one should expect the largest submatrix to be "skinny", that is, a few columns and lots of rows, or vice versa. For example, we expect the largest submatrix in a random $\{0,1\}$-matrix of size 256×256 to be size 2×160 (see Table 2).

5 Implementation and Experiments

We implemented column- and row-sampling algorithms in C++ and tested the programs on a desktop PC with a 1.2GHz Athlon CPU and 1GB of RAM, under Linux. Although the algorithms do not require sophisticated data structures, in order to carry out step 4 in the algorithm of Figure 2, one needs a data structure to store the strings and their frequencies. Since k and a are usually not very large, our experience shows that a simple hash table (of size a^k) is a good choice. If a^k becomes too large, a trie would be a better data structure. If one uses the hash table, it is important to keep track of the non-zero entries in another balanced data structure. That would avoid the algorithm to spend $O(a^k)$ to search for the frequently occurring strings. Observe also that row-sampling algorithm does not require any hash table, or any other data structure. However, our experiments show that in order to get the same level of performance of the column sampling, the row sampling strategy needs a significantly larger projection k which adversely affects the running time.

Another issue is whether one should keep track of the projections generated so far to avoid generating duplicates. We studied this matter experimentally, and found that it is worthwhile to keep track of the projections in some balanced data structure only when k is small. If k is large, the overhead required to keep the data structure updated is much higher than the time wasted in processing the same projection multiple times.

5.1 Simulations

In order to evaluate the performance of the algorithms, we designed several simulation experiments. In these experiments we randomly generated one thousand

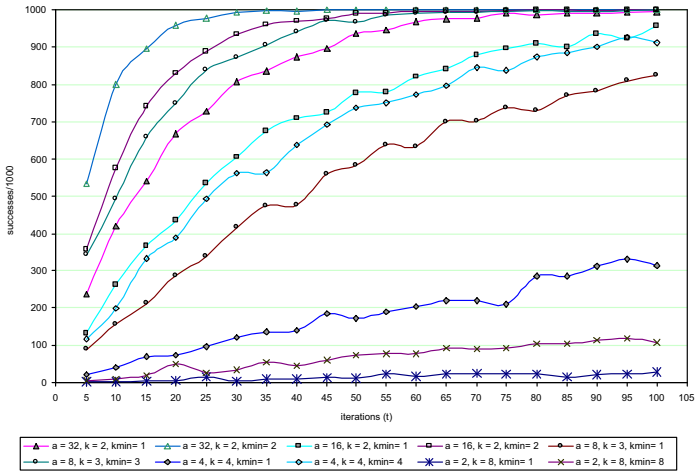

Fig. 3. Comparing the performance of the randomized algorithm LARGEST_BICLUSTER_C when $k_{\min} = k$ versus $k_{\min} = 1$, for different choices of the alphabet size a. The projection size is $k = \log_a m$

256×256 matrices of symbols drawn from an symmetric i.i.d. distribution over an alphabet of cardinality $a = 2, 4, 8, 16, 32$. Then, in each matrix we embedded a random 64×64 submatrix at random columns and random rows. We ran the algorithms for a few tens of iterations ($t = 5, \ldots, 100$), and for each choice of t we measured the number of successes out of the 1,000 distinct instances. Figure 3 summarizes the performance of LARGEST_BICLUSTER_C, for several choices of alphabet size a and projection size k, and minimum subset size $k_{\min}$. Figure 4 summarizes the performance of LARGEST_BICLUSTER_R under the same conditions.

In order to make a fair comparison between $k_{\min} = k$ and $k_{\min} = 1$, the number of iterations for the case $k_{\min} = k$ was multiplied by $2^k - 1$. Note that by doing so, we are assuming that one projection for $k_{\min} = 1$ takes about the same time as one projection for $k_{\min} = k$, which is not necessarily very accurate. Under this assumption, however, $k_{\min} = k$ outperforms $k_{\min} = 1$ (see Figure 3). This not necessarily true in the row sampling strategy (see Figure 4).

By comparing the performance of row sampling against column sampling, one can observe that if one uses the same set of parameters, column sampling always outperforms row sampling.

Unfortunately, we were unable to compare the performance of our randomized approach to other biclustering algorithms (e.g. [3, 4, 6, 5, 12, 8]), because their notion of bicluster is generally different from ours.

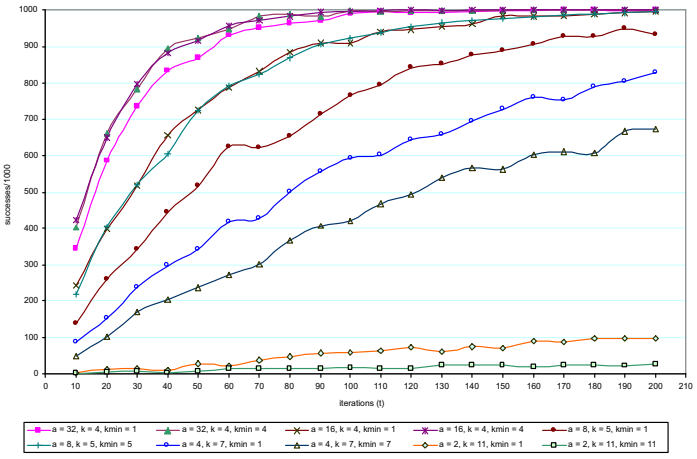

Fig. 4. Comparing the performance of the randomized algorithm LARGEST_BICLUSTER_R for different choices of the alphabet size a and projection size k

6 Conclusions

In this paper we have introduced the LARGEST BICLUSTER problem. This problem has a variety of applications ranging from computational biology to data mining. As far as we know, the pattern matching community has not looked yet at this problem from a combinatorial perspective. Unfortunately, the problem is generally NP complete.

Here we presented a rather simple algorithm based on random projections. Its performance with respect to the number of projection was carefully analyzed. We have also presented a probabilistic analysis of the LARGEST BICLUSTER problem, which allows one to determine the statistical significance of a solution.

Our approach performs remarkably well on synthetic data. On large alphabets, thirty or so iterations are enough to give a performance close to 100%. With respect to other biclustering algorithms (see e.g., [3, 12, 8]), our algorithm simultaneously discovers multiple solutions which satisfy the user-defined parameters without masking or changing the original data. In addition to this, the algorithm will never report solutions which are completely contained in other solutions.

Acknowledgments

This research was supported in part by NSF Grants CCR-0208709, DBI-0321756 and the NIH Grant R01 GM068959-01.

References

1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98). Volume 27,2 of ACM SIGMOD Record., New York, ACM Press (1998) 94–105
2. Aggarwal, C.C., Procopiuc, C., Wolf, J.L., Yu, P.S., Park, J.S.: Fast algorithms for projected clustering. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-99). Volume 28,2 of SIGMOD Record., New York, ACM Press (1999) 61–72
3. Cheng, Y., Church, G.M.: Biclustering of expression data. In: Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB-00), Menlo Park, CA, AAAI Press (2000) 93–103
4. Wang, H., Wang, W., Yang, J., Yu, P.S.: Clustering by pattern similarity in large data sets. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD-02), New York, ACM Press (2002) 394–405
5. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. In: Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology (ISMB'02), in Bioinformatics. Volume 18. (2002) S136–S144
6. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering local structure in gene expression data: The order-preserving submatrix problem. In: Proceedings of Sixth International Conference on Computational Molecular Biology (RECOMB 2002), ACM Press (2002) 45–55
7. Zhang, L., Zhu, S.: A new clustering method for microarray data analysis. In: Proceedings of the First IEEE Computer Society Bioinformatics Conference (CSB'02), IEEE Press (2002) 268– 275
8. Sheng, Q., Moreau, Y., Moor, B.D.: Biclustering microarray data by Gibbs sampling. In: Proceedings of European Conference on Computational Biology (ECCB'03). (2003) to appear
9. Mishra, N., Ron, D., Swaminathan, R.: On finding large conjunctive clusters. In: Proc. of the ACM Conference on Computational Learning Theory (COLT'03). (2003) to appear
10. Procopiuc, M., Jones, M., Agarwal, P., Murali, T.M.: A Monte-Carlo algorithm for fast projective clustering. In: Proceedings of the 2002 International Conference on Management of Data (SIGMOD'02). (2002) 418–427
11. Hartigan, J.A.: Direct clustering of a data matrix. Journal of the American Statistical Association **67** (1972) 123–129
12. Murali, T.M., Kasif, S.: Extracting conserved gene expression motifs from gene expression data. In: Proceedings of the Pacific Symposium on Biocomputing (PSB'03). (2003) 77–88
13. Hochbaum, D.S.: Approximating clique and biclique problems. Journal of Algorithms **29** (1998) 174–200
14. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York, NY (1979)
15. Grigni, M., Manne, F.: On the complexity of the generalized block distribution. In: In proceedings of Irregular'96, The third international workshop on parallel algorithms for irregularly structured problems. Volume 1117., Lecture Notes in Computer Science, Springer (1996) 319–326

16. Peeters, R.: The maximum-edge biclique problem is NP-complete. Technical Report 789, Tilberg University: Faculty of Economics and Business Adminstration (2000)
17. Dawande, M., Keskinocak, P., Swaminathan, J.M., Tayur, S.: On bipartite and multipartite clique problems. Journal of Algorithms **41** (2001) 388–403
18. Pasechnik, D.V.: Bipartite sandwiches. Technical report, available at `http://arXiv.org/abs/math.CO/9907109` (1999)
19. Reinert, G., Schbath, S., Waterman, M.S.: Probabilistic and statistical properties of words: An overview. J. Comput. Bio. **7** (2000) 1–46
20. Hastie, T., Tibshirani, R., Eisen, M., Alizadeh, A., Levy, R., Staudt, L., Chan, W., Botstein, D., Brown, P.: 'Gene shaving' as a method for identifying distinct sets of genes with similar expression patterns. Genome Biol. **1** (2000) 1–21
21. Lazzeroni, L., Owen, A.: Plaid models for gene expression data. Statistica Sinica **12** (2002) 61–86
22. Kluger, Y., Basri, R., Chang, J., Gerstein, M.: Spectral biclustering of microarray data: coclustering genes and conditions. Genome Res. **13** (2003) 703–16
23. Yang, J., Wang, H., Wang, W., Yu, P.S.: Enhanced biclustering on gene expression data. In: IEEE Symposium on Bioinformatics and Bioengineering (BIBE'03). (2003) to appear
24. Hanisch, D., Zien, A., Zimmer, R., Lengauer, T.: Co-clustering of biological networks and gene expression data. In: Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB-02), AAAI Press (2002) 145–154
25. Dhillon, I., Mallela, S., Modha, D.: Information-theoretic co-clustering. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-03), ACM Press (2001) to appear
26. Gu, M., Zha, H., Ding, C., He, X., Simon, H.: Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering. Technical Report CSE-01-007, Department of Computer Science and Engineering, Pennsylvania State University (2001)
27. Dhillon, I.: Co-clustering documents and words using bipartite spectral graph parititioning. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-01), New York, ACM Press (2001) 269–274
28. Szpankowski, W.: Average Case Analysis of Algorithms on Sequences. Wiley Interscience (2001)