APPLICATION OF BICLUSTERING ALGORITHMS TO BIOLOGICAL DATA

MASTERS THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of the Ohio State University

By
Kemal Eren, BS
Graduate Program in Computer Science

The Ohio State University
2012

Thesis Committee:
Ümit V. Çatalyürek, Advisor
Srinivasan Parthasarathy
Microarrays have made it possible to cheaply collect large gene expression datasets. Biclustering has become established as a popular method for mining patterns in these datasets. Biclustering algorithms simultaneously cluster rows and columns of the data matrix; this approach is well suited to gene expression data because genes are not related across all samples, and vice versa. In the past decade many biclustering algorithms that specifically target gene expression data have been published. However, only a few are commonly used in bioinformatics pipelines. There are a few reasons for this omission: implementations for only a small fraction of these algorithms have been published. Those that have been published have different interfaces, and there are few comparisons of algorithms or guidelines for choosing among them in the literature.

In this thesis we address three problems: the development of an efficient and effective biclustering algorithm, the development of a software framework for biclustering tasks, and a comprehensive benchmark of biclustering techniques.

We improved the Correlated Patterns Biclustering (CPB) algorithm’s running time and accuracy by modifying its heuristic for evaluating rows and columns for inclusion in a bicluster. This calculation was previously performed by an iterative approach, but we developed a more computationally efficient method. We further improved CPB by removing unnecessary parameters and developing a nonparametric method for filtering irrelevant biclusters.
To provide a common interface and also enable comparison of biclustering algorithms, we developed a Python package for bicluster analysis, which we introduce in this thesis. This package, BiBench, provides wrappers to twelve biclustering algorithms, as well as functionality for generating synthetic data, downloading gene expression data, transforming datasets, and validating biclusters.

Using BiBench we compared twelve algorithms, including the modified version of CPB. The algorithms were tested on synthetic datasets for their ability to recover specific bicluster models, resist noise, recover multiple biclusters, and recover overlapping biclusters. They were also tested on gene expression data; gene ontology enrichment was used to identify biologically relevant biclusters.
For Susan Slattery
ACKNOWLEDGMENTS

This work would have been impossible without the help of the people, both past and present, at the HPC Lab. In particular I would like to acknowledge my adviser, Ümit Çatalyürek, for all of his guidance and support.
VITA

1986 ............................... Born in Cleveland, OH.

2009 ............................... B.S. in Biology, minors in Mathematics and Computer Science, The University of Michigan.

2009-Present ....................... Joined the High Performance Computing Laboratory, The Ohio State University.

FIELDS OF STUDY

Major Field: Computer Science and Engineering

Specialization: Machine Learning, Data Mining, Bioinformatics
# TABLE OF CONTENTS

Abstract ................................................................. ii  
Dedication ................................................................. iii  
Acknowledgments ......................................................... v  
Vita ........................................................................ vi  
List of Figures .............................................................. ix  
List of Tables ............................................................... xi  

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
</table>
| 1       | Introduction ........................................ 1  
2       | Background ........................................... 5  
 | 2.1 Notation ................................................. 7  
 | 2.2 Algorithms ............................................... 8  
 | 2.2.1 Cheng and Church .................................... 8  
 | 2.2.2 OPSM ................................................... 9  
 | 2.2.3 xMOTIFs ............................................... 10  
 | 2.2.4 Spectral ............................................... 10  
 | 2.2.5 ISA ..................................................... 10  
 | 2.2.6 Plaid ................................................... 11  
 | 2.2.7 BiMax ................................................... 11  
 | 2.2.8 Bayesian Biclustering (BBC) ....................... 11  
 | 2.2.9 QUBIC ............................................... 12  
 | 2.2.10 COALESCE ........................................... 12  
 | 2.2.11 FABIA ............................................... 12  
3       | The Correlated Patterns Biclustering Algorithm .... 14  
 | 3.1 Original algorithm ....................................... 17  
 | 3.1.1 The tendency vector ................................... 18  
 | 3.1.2 Updating rows of a bicluster ....................... 23  

vii
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Illustration of a bicluster made contiguous by row and column exchanges.</td>
</tr>
<tr>
<td>2.1</td>
<td>Three types of bicluster structures. Note that biclusters may not be continuous, as in these figures, until the appropriate row and column permutation.</td>
</tr>
<tr>
<td>3.1</td>
<td>Effect of noise on correlation of rows of ten 50x50 matrices with perfectly correlated rows. Top: pairwise correlation density. Bottom: correlation density with tendency vector. 'Random' is a control for comparison: randomly generated matrices with no special correlation between rows.</td>
</tr>
<tr>
<td>3.2</td>
<td>Relationship between CORR and ERROR on random vectors with 50 elements.</td>
</tr>
<tr>
<td>3.3</td>
<td>Row expression levels before and after making all pairwise correlations positive.</td>
</tr>
<tr>
<td>3.4</td>
<td>Running time: iterative vs single-pass tendency vector calculation.</td>
</tr>
<tr>
<td>3.5</td>
<td>Scores: iterative vs single-pass tendency vector calculation.</td>
</tr>
<tr>
<td>3.6</td>
<td>The cone formed by tendency vector ( t ) and angle ( \theta = \arccos(\rho_{\text{min}}) ).</td>
</tr>
<tr>
<td>3.7</td>
<td>( \rho_{\text{min}} ) as a function of ( \rho_{\text{pw}} ).</td>
</tr>
<tr>
<td>3.8</td>
<td>Effect of row-to-column ratio on CPB’s scores for three different bicluster sizes: 25, 50, and 100 rows. (Number of bicluster columns were 100, 50, and 25, so that each bicluster always contained 2500 elements.</td>
</tr>
<tr>
<td>3.9</td>
<td>Improvement of relevance scores after filtering.</td>
</tr>
<tr>
<td>3.10</td>
<td>Running times of original C version and Python version of CPB as the number of rows in the dataset increases.</td>
</tr>
</tbody>
</table>
4.1 Four types of visualization supported by BiBench. The first three are performed by the biclust package; the last by BicOverlapper. 41

5.1 The intersection area (dark red) and union area (all shaded areas) involved in the calculation of the Jaccard index. 49

5.2 Bicluster model experiment. Each data point represents the mean of twenty datasets. A score of (1,1) is best. 51

5.3 Results of bicluster model experiment after filtering. 52

5.4 Noise experiments: additive noise. Dots shows the mean score; lines show one standard deviation. 54

5.5 Number experiment: increasing number of biclusters. 56

5.6 Overlap experiment: increasing amounts of overlap between two clusters. 58

5.7 Number of rows in all GDS biclusters. 62

5.8 Number of rows in enriched GDS biclusters. 63

5.9 Number of rows in enriched, non-overlapping GDS biclusters. 64
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Theoretical support of each bicluster model by algorithm.</td>
</tr>
<tr>
<td>5.2</td>
<td>GDS datasets</td>
</tr>
<tr>
<td>5.3</td>
<td>Aggregated results on all eight GDS datasets. Biclusters were considered enriched if any GO term was enriched with $p = 0.05$ level after multiple test correction. The set of enriched biclusters was further processed to discard overlapping biclusters.</td>
</tr>
<tr>
<td>5.4</td>
<td>Five most enriched terms for each algorithm’s best bicluster on GDS589.</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Clustering refers to a class of unsupervised methods for discovering relationships between data elements. The most popular methods partition a set of \( n \) objects with \( m \) features into a \( k \) clusters, with the goal of minimizing between-cluster similarity and maximizing between-cluster similarity given some similarity metric. However, there are many other formulations of clustering problems. This thesis focuses on a model of clustering known as biclustering, which simultaneously clusters both the \( n \) objects and their \( m \) features. Unlike traditional clustering methods, which consider all features, biclustering algorithms can discover local patterns on only a subset of features. If the \( n \) objects are arranged into an \( m \) by \( n \) feature-object matrix, a bicluster consists of a subset of the \( m \) rows and a subset of the \( n \) columns. After row and column exchanges, the rows and columns of the bicluster define a contiguous submatrix of the data matrix; see Figure 1.1.

To demonstrate how biclustering might be used, consider exploring the following dataset: a corpus \( D \) of documents, with \( |D| = n \). Clustering the documents might yield clusters corresponding to, i.e., their subjects. However, this approach does not reveal local similarity between documents. We may extract more information from \( D \) by applying biclustering to the problem as follows. Construct a word occurrence matrix \( M \) with \( m \) rows and \( n \) columns. Each row represents a word that appears in at least one of the documents, each column represents a document, and \( M_{ij} \) contains
the number of times word $i$ appeared in document $j$. After biclustering $M$, each bicluster not only contains similar documents but also the words over which they are similar.

Just as the corpus $D$ was converted into data matrix $M$, any data that can be coerced into a matrix may be biclustered. This flexibility makes biclustering a useful approach for mining data with local patterns in many fields. In particular, biclustering has become popular for exploring gene expression data.

Gene expression data, or microarray data, contains measurements of the expression levels of genes. A microarray chip contains a grid of $m$ probes, each of which hybridize with a particular cDNA or mRNA sequence. A single microarray experiment generates a vector of $m$ expression measurements in parallel. Repeating for multiple samples yields a $m$ by $n$ matrix of data. Microarray technology has enabled the creation of large databases of gene expression data, which may be mined for biologically relevant patterns, such as previously unknown gene relationships.

Unsupervised methods such as clustering and biclustering are an important first step in analyzing these data sets. Exploring all possible relationships between genes would be prohibitively expensive; clustering algorithms help mitigate this expense
by highlighting possible genetic interactions and guiding the development of followup
the relationships between genes often manifest in local patterns that are undetectable
by other clustering algorithms. For instance, two gene expression profiles may not
be related in all samples, because those genes may interact only in specific environ-
ments. Similarly, two samples almost certainly do not show similar expression levels
across all genes. Because of these advantages, many biclustering algorithms have
been developed specifically for gene expression data.

Though the popularity of biclustering has led to the publication of many algo-
rithms, there are still many open lines of research. The landscape of possible bicluster
models is relatively unexplored; in particular it is not clear which models produce the
most biologically relevant biclusters. Many new algorithms are possible, and existing
algorithms may be modified to improve their results. New methods are published
frequently, so the number of available algorithms from which to choose is expanding.
As the number of available methods grows, the problem of selecting the appropriate
algorithm for a data mining task becomes increasingly difficult.

Some comparisons, both theoretical and empirical, have been published (see Sec-
tion 5.1), but the task of comparing biclustering algorithms is far from complete.
There are many factors that make such comparisons difficult. Theoretical difficulties
include the variety of biclustering models, many of which are not directly compar-
able. Even for algorithms with similar models, the choice of comparison criteria is
not obvious and may depend on the specific dataset under consideration. Empirical
difficulties include the lack of implementations of algorithms; those implementations
that do exist are written in different languages with incompatible interfaces.

In this thesis we address three problems: the development of an efficient and effect-
tive biclustering algorithm, the development of a software framework for biclustering
tasks, and a comprehensive benchmark of biclustering techniques.
In Chapter 3 we describe our modifications to the Correlated Patterns Biclustering (CPB) algorithm to improve its accuracy and running time [4]. CPB seeks biclusters with high pairwise correlations between genes. To evaluate rows for membership in a bicluster it calculates a heuristic using an iterative method. We replaced this iterative method with a linear time method, significantly improving CPB’s running time. In addition to being asymptotically faster, the new method also improves CPB’s accuracy and resistance to noise.

To make bicluster analysis more convenient we developed BiBench, a Python package that provides wrappers to twelve biclustering algorithms. BiBench also provides functionality for generating synthetic data, downloading gene expression data, transforming datasets, and validating biclusters. In Chapter 4 we introduce BiBench and review its main features.

Chapter 5 formulates similarity measures for biclusters and describes a comprehensive comparison of biclustering algorithms across a suite of tests. Though we anticipate that BiBench will make many data analysis tasks easier, it was originally conceived to aid in writing experiments for comparing existing biclustering algorithms. We used BiBench to compare twelve algorithms, including the modified version of CPB, on both synthetic data and gene expression data. Similar comparisons used only one bicluster data model, biasing results against algorithms that do not fit that model. Instead we used six different bicluster models: constant, upregulated, plaid, shift, scale, and shift-scale models. Synthetic datasets were used to test each algorithms for their ability to recover each model, resist noise, recover multiple biclusters, and recover overlapping biclusters. Each algorithm was also used to bicluster data from the Gene Expression Omnibus; results were evaluated using Gene Ontology enrichment analysis.

Finally, Chapter 6 concludes and discusses further avenues for research.
In this chapter we give a brief introduction to the biclustering problem and describe the algorithms compared in later chapters. Because of the variety of approaches to biclustering, a fully general description is difficult to formulate, and a full classification of all published biclustering methods is beyond the scope of this thesis. For more complete surveys of biclustering, see Madeira and Oliveira [33], Tanay et al. [49], Busygin et al. [6], and Fan et al. [16]. In statistics related methods are often called two-way clustering or co-clustering; for surveys see Van Mechelen et al. [51], Patrikainen and Meila [36], Yoon et al. [56], and Kriegel et al. [30].

All biclustering algorithms may be conceptually divided into two components: a bicluster model which specifies the clustering task, and a search strategy for finding solutions fitting that model. We first address the major differences among model formulations, then the approaches used for efficiently finding solutions.

Biclustering algorithms differ widely in their exact problem formulation. The main difference between methods is whether they are defined in terms of data modeling or bicluster recognition. Modeling approaches attempt to model the data matrix, often by fitting a statistical model or set of generative parameters. The final biclusters may be extracted from the fitted model, with the fitness of a solution defined as some measure of the error between the fitted data and the true data matrix. The Plaid and FABIA models are two examples of data modeling approaches; see Sections 2.2.6 and
Figure 2.1: Three types of bicluster structures. Note that biclusters may not be continuous, as in these figures, until the appropriate row and column permutation.

2.2.11. In contrast, recognition methods define the fitness of individual biclusters, or sets of biclusters, without modeling the overall dataset. The fitness of such biclusters is defined in terms of some local criteria, such as homogeneity or coherency, computed only on the bicluster submatrix. Most of the algorithms examined in this chapter use this approach.

Not all biclustering methods define biclusters in the same way. In one common structure, the rows of a bicluster share some relationship across the columns of the bicluster, but not the remaining columns, and the columns of the bicluster share some relationship across rows of the bicluster, but not the remaining rows. In other words, each row and each column of the data matrix is assigned to a single cluster; see Figure 2.1(a). Another common structure partitions the dataset into a checkerboard pattern, seeking tiles that meet some fitness criterion; see Figure 2.1(b). A third
common structure defines a bicluster as any submatrix with some fitness property; see Figure 2.1(c). Of course, other structures are also possible.

Given a biclustering model, the problem becomes efficiently finding optimal solutions. Most such tasks are exponential in the rows and columns of the matrix. The proof of the exact computational complexity depends on the model, but it is easy to show that even a simple biclustering task is complex. For instance, consider the task of finding the largest bicluster of ones in an $m \times n$ binary dataset. Exhaustively evaluating all biclusters requires iterating through all $2^m$ subsets of rows and all $2^n$ subsets of columns. Therefore the total number of candidate biclusters to consider is $2^{mn}$. To compensate, most optimization strategies use approximations, heuristics, and randomization to short-circuit the running time. These strategies include matrix decomposition such as singular value decomposition, greedy iterative approaches, mining of bipartite graph models, and Metropolis-Hastings methods [33].

2.1 Notation

Throughout this thesis we use the following vector, matrix, and set notation:

- Sets are denoted by uppercase cursive letters: $\mathcal{B}$, $\mathcal{R}$, $\mathcal{C}$.
- When necessary to distinguish sets of sets, they are denoted by blackboard capitals: $\mathbb{B}$.
- Scalars are denoted by lowercase letters: $x$, $y$, $z$.
- Row and column vectors are denoted by lowercase bold letters: $\mathbf{x}$, $\mathbf{y}$, $\mathbf{t}$.
- Element $i$ of vector $x$ is denoted $x[i]$ or $x_i$.
- Matrices are denoted by uppercase bold letters: $\mathbf{M}$, $\mathbf{X}$, $\mathbf{Y}$.
Element \( i, j \) of matrix \( M \) is denoted \( M[i, j] \) or \( M_{ij} \).

- Row vector \( i \) of matrix \( M \) is denoted \( M[i, \ast] \) or \( M_{i\ast} \).

- Column vector \( j \) of matrix \( M \) is denoted \( M[\ast, j] \) or \( M_{\ast j} \).

- Subsets of matrices are denoted by a set in the row and/or column position of the matrix notation. Let \( M \) be an \( m \times n \) matrix, \( R = \{1, 3, 5\} \), and \( C = \{2, 4\} \). Then:
  - \( M[R, \ast] \) is a \( 3 \times n \) submatrix of \( M \).
  - \( M[\ast, C] \) is an \( m \times 2 \) submatrix of \( M \).
  - \( M[R, C] \) is a \( 3 \times 2 \) submatrix of \( M \).
  - row vector \( M[i, C] \) is a \( 1 \times 2 \) submatrix of \( M \).
  - column vector \( M[R, j] \) is a \( 3 \times 1 \) submatrix of \( M \).

2.2 Algorithms

Biclustering was first developed by Hartigan [22], but it was not applied to gene expression data until Cheng and Church [7]. In this thesis we focus on twelve algorithms, the implementations for all of which are publicly available. A complete description of one of the algorithms, CPB, appears in Chapter 3; the others are briefly summarized in this section.

2.2.1 Cheng and Church

Named for its authors, this algorithm was the first to be applied to gene expression data [7]. It is a deterministic greedy algorithm that seek to find the biclusters with low variance, as defined by a criterion known as the mean squared residue (MSR),
which is calculated as follows. If $R$ and $C$ are the sets of rows and sets of columns of the bicluster $B$ respectively, the Residue value of a single element of $B$ is calculated as:

$$\text{Residue}(r, c) = X_{rc} - X_{rC}^* - X_{RC}^* + X_{RC}^*$$ (2.1)

where $X_{rC}^*$ is the mean of row $r$, $X_{RC}^*$ is the mean of column $c$, and $X_{RC}^*$ is the mean of the entire bicluster. Then the MSR value for the entire bicluster is calculated as the sum of squared residues, normalized by the size of the bicluster:

$$\text{MSR}(B) = \frac{1}{|R||C|} \sum_{r \in R, c \in C} (\text{Residue}(r, c))^2$$ (2.2)

To find these biclusters, the algorithm starts with the whole data matrix, removing the rows and the columns that have high residues. Once the mean squared residue of the bicluster reaches a given $\delta$ parameter, the rows and columns with smaller residue than the bicluster residue are added back to the bicluster. If multiple biclusters are to be recovered, the found biclusters are masked with random values, and the process repeats.

The speed and accuracy tradeoff of Cheng and Church is controlled by a parameter $\alpha$, which controls pruning.

### 2.2.2 OPSM

The Order-Preserving Submatrix (OPSM) algorithm is a deterministic greedy algorithm that seeks biclusters with ordered rows [2]. The OPSM model defines a bicluster as an order-preserving submatrix, in which there exists a linear ordering of the columns in which the expression values of all rows of that submatrix are strictly increasing from left to right. OPSM constructs complete biclusters by iteratively growing partial biclusters, scoring each by the probability that it will grow to some fixed target size. Only the best $l$ partial biclusters are kept at each iterations.
2.2.3 xMOTIFs

xMOTIFs is a nondeterministic greedy algorithm that seeks biclusters with conserved rows in discretized dataset [35]. For each row, the intervals of the discretized states are determined according to the statistical significance of the interval compared to the uniform distribution. For each randomly selected column, called a seed, and for each randomly selected set of columns, called discriminating sets, xMOTIFs tries to find rows that have same states over the columns of the seed and the discriminating set. Therefore, xMOTIFs can find biclusters with constant values and with constant rows.

2.2.4 Spectral

Spectral uses singular value decomposition to find a checkerboard pattern in the data in which each bicluster is up- or down-regulated [29]. Only biclusters with variance lower than a given threshold are returned.

2.2.5 ISA

The Iterative Signature Algorithm is a nondeterministic greedy algorithm that seeks biclusters with two symmetric requirements: each column in the bicluster must have an average value above some threshold $T_C$; likewise each row must have an average value above some threshold $T_R$ [3]. The algorithm starts with a seed bicluster consisting of randomly selected rows. It iteratively updates the columns and rows of the bicluster until convergence. By re-running the iteration step with different row seeds, the algorithm finds different biclusters. ISA can find upregulated or downregulated biclusters.
2.2.6 Plaid

The Plaid algorithm was published as an improvement to the algorithm published in Lazzeroni and Owen [31], which introduced the generative model known as the plaid model Turner et al. [50]. The plaid model is an additive model in which a data element $X[r, c]$, with $K$ biclusters assumed present, is generated as the sum of a background effect $\theta$, cluster effects $\mu$, row effects $\alpha$, column effects $\beta$, and random noise $\epsilon$:

$$X[r, c] = \theta + \sum_{k=1}^{K} (\mu_k + \alpha_{rk} + \beta_{ck}) \rho_{rk} \kappa_{ck} + \epsilon_{rc} \quad (2.3)$$

where the background refers to any matrix element that is not a member of any bicluster. The Plaid algorithm fits this model by iteratively updating each parameter of the model to minimize the mean squared error between the modeled data and the true data.

2.2.7 BiMax

BiMax is a simple reference algorithm that seeks biclusters of 1’s in a binary matrix; it was introduced as a reference algorithm for algorithm comparisons Prelić et al. [40]. It biclusters the data matrix in a divide-and-conquer approach, recursively dividing it into a checker board format. Because the algorithm works only on binary data, datasets must first be converted to a binary representation, or binarized. The method used for binarization has a large influence on BiMax’s performance, since it determines which data elements may be part of a bicluster.

2.2.8 Bayesian Biclustering (BBC)

The Bayesian BiClustering algorithm uses Gibbs sampling to fit a hierarchical Bayesian version of the plaid model [21]. It restricts overlaps to occur only in rows or columns,
not both, so that two biclusters may not share the same data elements. The sampled posteriors for cluster membership of each row and column represent fuzzy membership; thresholding yields crisp clusters.

2.2.9 QUBIC

QUBIC is a deterministic algorithm that reduces the biclustering problem to finding heavy subgraphs in a bipartite graph representation of the data [32]. It seeks biclusters with nonzero constant columns in discrete data. The data is first discretized into down and upregulated ranks, then biclusters are generated by iterative expansion of a seed edge. The first expansion step requires that all columns be constant; in the second step this requirement is relaxed to allow the addition of rows that are not totally consistent.

2.2.10 COALESCE

COALESCE is a nondeterministic greedy algorithm that seeks biclusters representing genetic regulatory modules [26]. This algorithm can find upregulated and downregulated biclusters. It begins with a random set of genes, then iterates, updating columns and rows until convergence. It select columns by two-population z-test, motifs by a modified z-test, and then selects rows by posterior probability. COALESCE can use gene sequence data.

2.2.11 FABIA

FABIA fits a generative multiplicative model [24]. It models the $m$ by $n$ data matrix $X$ as the sum of $k$ biclusters plus additive noise $\Upsilon$, where the $i^{th}$ bicluster is the outer product of two sparse vectors: a row vector $\lambda$ and a column vector $z^T$, where $|\lambda| = m$ and $|z| = n$. Collecting the row vectors into a $k$ by $m$ matrix $\Lambda$ and the column
vectors into a $k$ by $n$ matrix $Z$, the gene expression data matrix $X$ is modeled as follows:

$$X = \sum_{i=1}^{k} \Lambda_{is} Z_{is}^T + \Upsilon = \Lambda Z + \Upsilon$$  \hspace{1cm} (2.4)

Two factor analysis models are used to fit this model to the data set; variational expectation maximization is used to maximize the posterior. Row and column membership in each bicluster is fuzzy, but thresholds may be used to make crisp clusters.
CHAPTER 3

THE CORRELATED PATTERNS BICLUSTERING ALGORITHM

Correlated Patterns Biclustering (CPB), originally developed by Bozdağ et al. [4], is a biclustering algorithm designed specifically to find sets of genes that are related to some target gene of interest. In this chapter we introduce the original algorithm and formulate some improvements that both speed up its running time and improve its accuracy.

CPB seeks biclusters in which genes expression profiles are correlated, or linearly related. Not only are these relationships likely to be biologically relevant, but many other patterns frequently sought by biclustering algorithms are special cases of correlation. Pearson’s correlation coefficient measures the strength of the linear relationship between pairwise data elements \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\). If these points are all solutions to the equation \(y = ax + b\) for some \(a, b \in \mathbb{R}\), the data are perfectly linearly correlated. If \(a > 0\), they are positively correlated and achieve a maximum correlation of 1; otherwise they are negatively correlated and achieve a minimal correlation of \(-1\). A correlation of 0 implies no linear relationship between \(x\) and \(y\).

After centering each vector so that its mean is zero, this relationship may be
written as as the inner product of the vectors divided by the product of their norms. So if we let \( \hat{x} = x - \overline{x} \) and \( \hat{y} = y - \overline{y} \), it becomes:

\[
\text{Corr}(x, y) = \frac{\hat{x} \cdot \hat{y}}{||\hat{x}|| \cdot ||\hat{y}||} \quad (3.1)
\]

But expressing correlation in this fashion makes it clear that it is simply the cosine of the angle between the two vectors \( \hat{x} \) and \( \hat{y} \):

\[
\text{Corr}(x, y) = \cos(\theta_{\hat{x}, \hat{y}}) \quad (3.2)
\]

This way of interpreting PCC makes many of its properties clear. \( x \) and \( y \) are correlated when the angle between \( \hat{x} \) and \( \hat{y} \) is small: \( \cos(0) = 1 \); they are uncorrelated when the vectors are orthogonal: \( \cos(\pi/2) = 0 \); they are negatively correlated when their angle is maximized: \( \cos(\pi) = -1 \). It is also clear that the correlation between two vectors is invariant to translation and scaling, so that for \( a, b, c, d \in \mathbb{R} \):

\[
\text{Corr}(x, y) = \text{Corr}(a + bx, c + dy) \quad (3.3)
\]

PCC emerges in many statistical methods, such as linear regression. Because PCC is invariant to translation and scaling, biclustering algorithm seeking biclusters with correlated rows would theoretically be capable of discovering so-called shift-scale bicluster patterns. We will use both properties – invariance to shifting/scaling and the interpretation of correlation as the cosine of angles – in the development of an efficient biclustering method to find correlated genes.

Regulatory relationships between genes may be expressed as linear correlations in their expression profiles. For instance, the expression levels of two genes that are positively regulated by the same upstream factor are likely to be positively correlated (high when the factor is present and low otherwise) whereas if one of the genes is inhibited by that factor, they are likely to be negatively correlated. However, gene
regulatory networks are complex and interrelated, so simply examining all pairwise correlations of rows in a gene expression data matrix is insufficient, because genes may be highly correlated only on a subset of samples. Moreover, PCC is not transitive, so mining pairwise correlations for larger sets of related genes is not trivial. Therefore, we attempt to find these gene relationships by biclustering.

There are many possible ways to define biclusters. In this chapter we are interested in finding local patterns, and so we define biclusters as follows:

**Definition 1 (Bicluster).** Let $M$ be an $m$ by $n$ matrix. A bicluster $B = (R, C)$ of $M$ is a subset of rows $R \in \text{RANGE}(m)$ and a subset of columns $C \in \text{RANGE}(n)$ of the data matrix.

where $\text{RANGE}(x)$ is simply the first $x$ natural numbers:

**Definition 2 (Range).** $\text{RANGE}(x) = \{i : i \in \mathbb{N}, 1 \leq i \leq x\}$.

Then a bicluster $(R, C)$ naturally defines a subset of the data matrix: $M[R, C]$.

Each bicluster exists independently of any others, and therefore there may be many, possibly overlapping, biclusters in a single dataset. This definition is appropriate for finding local patterns, but not for fitting a global model to the data; other formulations of biclusters exist for such tasks.

We are interested in finding biclusters in a data matrix $M$ such that the minimal pairwise correlation between any two rows in the bicluster is larger than a given threshold $\rho$. In other words, the fitness of a bicluster $(R, C)$ is calculated as $\text{FITNESS}(M[R, C])$, where $\text{FITNESS}(X)$ is the function that takes a matrix $X$ and calculates the minimum correlation between any two rows in $X$:

$$\text{FITNESS}(X) = \min_{i<j} \text{CORR}(X[i, *], X[j, *])$$

Finding all possible biclusters $(R, C)$ in the $m$ by $n$ matrix $M$ that have $F(M[R, C]) \geq \rho$ for some correlation threshold $\rho$ would require examining all possible subsets of rows
and columns, and therefore the task is in $O(2^{nm})$. Instead, we have developed an iterative process, Correlated Patterns Biclustering (CPB), that finds such biclusters much more efficiently.

### 3.1 Original algorithm

Before discussing the improvements we made to CPB, we first outline the main features of the algorithm as published.

Algorithm 1 outlines the steps involved in the CPB algorithm. The process starts with a randomly generated initial bicluster $B = (\mathcal{R}, \mathcal{C})$. This initial seed is improved iteratively by alternating moving rows and columns in and out of the bicluster using a search technique similar to mean-shift [9]. To target a specific gene of interest, CPB may take as a parameter $r_{ref}$, a reference row which is included in the initial seed and forced to remain in the bicluster in each successive iteration.

There are two main loops in the algorithm. The inner loop refines the bicluster, while the outer loop repeats this refinement process, each time updating the parameters $\rho_{\text{min}}$ and $\gamma_{\text{min}}$. The minimum correlation $\rho_{\text{min}}$ tightens, and the minimum number of rows $\gamma_{\text{min}}$ relaxes, thus sweeping a range of parameters while closing in on the final parameters.

To update rows, correlation is calculated between each row and a tendency vector $t = (t_1, \ldots, t_{|\mathcal{C}|})$ that represents general tendency of rows in the bicluster submatrix. If $\text{CORR}(M[r, \mathcal{C}], t)$ for a row $r$ is above a certain threshold, $r$ is included in set $\mathcal{R}$. On the other hand, using a similar criterion for columns is too restrictive. Instead, a good criterion for inclusion of a column $c$ into $\mathcal{C}$ should measure the impact of $c$ on the pairwise correlation of rows in $\mathcal{R}$. For this purpose, the ERROR function, discussed in Section 3.1.3 is used to evaluate similarity of tendencies of rows in $\mathcal{R}$ with respect to column $c$. 

17
In each inner iteration of CPB, first the tendency vector \( t \) for the current bicluster is computed. Then either the rows or the columns of the bicluster are updated, in an alternating fashion. Rows and columns are not updated simultaneously to avoid large fluctuations in the bicluster structure, which could slow down or prevent convergence. To update rows, \( R \) is updated to contain each row \( r \) that has \( \text{Corr}(M[r,C], t) \) above PCC threshold \( \rho_{\text{min}} \); the reference row \( r_{\text{ref}} \) is always included. To update columns, first the row \( r \) that has the smallest \( \text{Corr}(M[r,C], t) \) above threshold \( \rho_{\text{min}} \) is calculated. Then \( C \) is updated with each column \( c \) that has smaller error than row \( r \). The number of columns to include is modified by the minimum number of columns \( \gamma_{\text{min}} \). Iterations to update the bicluster end when neither \( R \) nor \( C \) changes after an iteration or after a certain iteration limit (the limit is usually 20; convergence is usually achieved in 5-10 iterations).

3.1.1 The tendency vector

CPB seeks biclusters such that all rows have some minimum correlation with all other rows. However, directly using the minimum pairwise row correlation would make a poor heuristic for iterative optimization, due to its instability. For a row \( r \) to be a member of set \( R \), it must be the case that \( \text{Corr}(M[r,C], M[i,C]) > \rho \) for all \( i \in R \). Directly requiring this condition while iteratively updating the bicluster is unstable, and can actually prevent convergence, because it allows a single row to dominate. For instance, it is very common for the random initial bicluster to contain some row that is not highly correlated with any other row. Even if every other row is highly correlated, this outlier controls the fitness of the whole bicluster, allowing no other row to enter.

Fortunately, there are properties of the correlation coefficient that allow us to build a more stable heuristic. As discussed previously, the correlation between two
Algorithm 1: Correlated Pattern Biclusters.

Data: $M_{mn}$, $\rho, \gamma, r_{ref}$
Result: Bicluster $B = (R, C)$.

begin
1 $R \leftarrow \text{RandSubset}(\text{Range}(m)) \cup \{r_{ref}\}$;
2 $C \leftarrow \text{RandSubset}(\text{Range}(n))$;
3 $\rho_{\text{min}} \leftarrow 0.9\rho$;
4 $\rho_{\Delta} \leftarrow (\rho - \rho_{\text{min}}) \frac{\text{outer iters}}{n}$;
5 $\gamma_{\min} \leftarrow |C|$;
6 $\gamma_{\Delta} \leftarrow (n - |C|) \frac{\text{outer iters}}{n}$;
7 for $i$ in $\{0 \ldots \text{outer iters}\}$ do // usually 5
8 for $j$ in $\{0 \ldots \text{inner iters}\}$ do // usually 20
9 $t \leftarrow \text{ComputeTendency}(M[R,C])$;
10 if $j$ mod 2 = 0 then
11 $R \leftarrow \text{UpdateRows}(M, R, C, t, \rho_{\text{min}}, r_{ref})$;
12 end
13 else
14 $C \leftarrow \text{UpdateCols}(M, R, C, t, \gamma_{\text{min}})$;
15 end
16 if $(R, C)$ has not changed then
17 break;
18 end
19 $\rho_{\text{min}} \leftarrow \rho_{\text{min}} + \rho_{\Delta}$;
20 $\gamma_{\text{min}} \leftarrow \gamma_{\text{min}} - \gamma_{\Delta}$;
21 end
22 return $B = (R, C)$
end

Algorithm 2: Compute tendency vector

Data: Matrix $X_{mn}$, $\Delta_{\text{min}}$
Result: $n$-length row vector $t$

begin
1 $t \leftarrow \text{ColMean}(X)$;
2 $\alpha, \beta \leftarrow$ intercept, slope of regressing $t$ on rows;
3 repeat
4 $\beta_{\text{old}} \leftarrow \beta$;
5 $t \leftarrow \text{ColMean}(X)$;
6 $\alpha, \beta \leftarrow$ intercept, slope of regressing $t$ on rows;
7 until $\max(|\beta - \beta_{\text{old}}|) < \Delta_{\text{min}}$;
8 return $t$;
9 end
Algorithm 3: Update Rows

Data: $M_{mn}, \mathcal{R}, \mathcal{C}, t, \rho_{\text{min}}, r_{\text{ref}}$
Result: new $\mathcal{R}$, with all rows correlated with $t$

1 begin
2 $R_{\text{new}} \leftarrow \{\}$;
3 for $r$ in RANGE$(m)$ do
4    if $\text{CORR}(M[r, \mathcal{C}], t) > \rho_{\text{min}}$ or $r = r_{\text{ref}}$ then
5        add $r$ to $R_{\text{new}}$;
6    end
7 end
8 return $R_{\text{new}}$
9 end

Algorithm 4: Update Columns

Data: $M_{mn}, \mathcal{R}, \mathcal{C}, t, \gamma_{\text{min}}$
Result: new $\mathcal{C}$

1 begin
2 $r \leftarrow \text{arg min}_{i \in \mathcal{R}}(\text{CORR}(M[i, \mathcal{C}], t))$;
3 $\alpha, \beta \leftarrow$ intercept and slope column vectors ;
4 $\epsilon \leftarrow \text{ERROR}(\frac{M[r, \mathcal{C}]-\alpha[r]}{\beta[r]}, t)$;
5 $t_{\text{ext}} \leftarrow \text{COLMEAN}((M[\mathcal{R}, *] - \alpha)/\beta)$;
6 $C \leftarrow \{c : c \in \text{RANGE}(n), \text{ERROR}(\frac{M[\mathcal{R}, c]-\alpha}{\beta}, t_{\text{ext}}[c]) > \epsilon\}$;
7 Add cols until $|C| > \gamma_{\text{min}}$;
8 return $C$
9 end
vectors is invariant to shifting and scaling. In fact, for any two vectors \( x \) and \( t \) that are exactly correlated, there exist shifting and scaling factors \( \alpha, \beta \in \mathbb{R} \) that exactly transform \( t \) into \( x \): \( x = \alpha + \beta t \). These factors are easily calculated with linear regression. If \( x \) and \( t \) are not exactly correlated, regressing \( t \) on \( x \) yields \( x \approx \alpha + \beta t \) when \( |\text{CORR}(x, t)| \) is large.

We denote \( t \) to be a tendency vector for \( x \), because \( t \) approximates \( x \) given the appropriate shifting and scaling factors. Now, given set of \( m \) row vectors in matrix \( X \) with \( n \) columns, we wish to find the single tendency vector \( t \) that best approximates each \( x \in X \). We do so using an iterative approach that takes advantage of the relationship between linear regression and Pearson’s \( r \). A first guess for \( t \) is calculated as \( \text{ColMean}(X) \).

**Definition 3 (ColMean).** \( \text{ColMean} \) is a function that takes an \( m \) by \( n \) matrix \( X \) and returns an \( n \)-length row vector \( x \), where \( x[i] \) is the mean of \( X \)’s \( i \)th column.

\[
\text{ColMean}(X) = [\text{Mean}(X[*], 1]), \text{Mean}(X[*], 2), \ldots, \text{Mean}(X[*], n)]
\]

This is more easily written using the following matrix-vector notation, where \( 1_m \) is the ones vector \([1 \ 1 \ \cdots \ 1]\) in \( \mathbb{R}^m \):

\[
\text{ColMean} = \frac{1}{m}(1_m \cdot X)
\]

Then slope and intercept for regressing \( t \) on each row are calculated using linear regression, yielding column vectors \( \alpha \) and \( \beta \). \( t \) is updated to \( \text{ColMean}((X - \alpha)/\beta) \), and the process iterates until convergence: when maximum change in slope is less than some threshold \( \Delta_{\text{min}} \). The algorithm for calculating the tendency vector of a matrix \( X \) is given in Algorithm 2.

Now given the tendency vector, we can construct a heuristic for evaluating a row for membership in the bicluster: instead of considering its pairwise correlation
Figure 3.1: Effect of noise on correlation of rows of ten 50x50 matrices with perfectly correlated rows. Top: pairwise correlation density. Bottom: correlation density with tendency vector. 'Random' is a control for comparison: randomly generated matrices with no special correlation between rows.
with all other rows in the bicluster, it is only necessary to consider its correlation
with the tendency vector. This heuristic does not suffer from the weakness of pair-
wise comparisons, because it reflects iterative improvements. Moreover, it is much
more resistant to noise: Figure 3.1 compares this heuristic to pairwise correlation
as Gaussian noise is added to 50x50 matrices with perfectly correlated rows. By
taking advantage of cluster-wide information, rather than considering only two rows
at a time, the tendency vector heuristic better captures the correlation relationship
between noisy rows.

### 3.1.2 Updating rows of a bicluster

The algorithm for updating rows is very simple: given a tendency vector \( t \) for the
current bicluster, the new bicluster includes all rows in the dataset that are correlated
with \( t \) by at least \( \rho_{\min} \). The pseudocode appears in Algorithm 3.

Algorithm 1 starts with a relaxed threshold \( \rho_{\min} \) and slowly tightens it in line 20
to the given threshold, \( \rho \). While tightening \( \rho_{\min} \), the constraint on minimum number
of columns, \( \gamma_{\min} \), is relaxed. This allows sweeping the search space between two
extreme combinations of these parameters. The initial values for \( \rho_{\min} \) and \( \gamma_{\min} \) are
set to 0.9\( \rho_{\min} \) and the number of columns in the initial bicluster, respectively.

To ensure that the reference row \( r_{\text{ref}} \) has a larger impact in decision mechanisms
of the algorithm, it is assigned a larger weight when computing the vector \( t \). The total
contribution from rows except \( r_{\text{ref}} \) is multiplied by \((1 - \omega)\) and the contribution from
\( r_{\text{ref}} \) is multiplied by \( \omega \), where \( \omega \) is an input parameter. Large values for \( \omega \) allows
discovering patterns that more closely resemble \( r_{\text{ref}} \); whereas small values reduce
sensitivity, hence offers higher tolerance to noise.
3.1.3 Updating columns of a bicluster

When updating columns of the bicluster, we want to include as many columns as possible while ensuring that each row’s correlation with the tendency vector does not decrease. Calculating the effect on the PCCs for every possible combination of columns is impractical. Instead, we use a method that allows us to consider each column individually. Our method ensures that we add as many columns as possible without decreasing the fitness of the bicluster; this method of updating while ensuring non-decreasing fitness is similar to that of the Information-Theoretic Co-Clustering algorithm [12].

We define the Error between two vectors as the norm of their difference, normalized by their cardinality:

**Definition 4 (Error).** \( \text{Error}(x_n, y_n) = \frac{1}{n}||x - y|| \)

and note that \( \text{Error}(x_n, y) \), the error between a vector and a scalar, is also defined as \( ||x_n - y|| \).

We assess the coherence of a column \( c \) across rows \( R \) by evaluating the error between \( M[R, c] \) and element \( c \) of an extended tendency vector \( t_{ext} \). The extended tendency vector \( t_{ext} \) is calculated as the tendency vector of \( M[R, \ast] \), but given the \( \alpha \) and \( \beta \) vectors already calculated for bicluster \( (R, C) \):

\[
t_{ext} = \text{ColMean}((M[R, \ast] - \alpha)/\beta)
\]

(3.5)

In CPB, only the columns \( c \) having \( \text{Error}(M[R, c], t_{ext}[c]) \) below a threshold \( \epsilon \) are included in the bicluster. \( \epsilon \) is is chosen to ensure that the included columns do not decrease the fitness of the bicluster. We choose \( \epsilon \) as follows. For a row \( r \in R \), \( r \)'s error is computed as its distance from the reference row, after transforming \( M[r, C] \) to match the reference row as closely as possible: \( \text{Error}((M[r, C] - \alpha[r])/\beta[r], t) \). Then
Figure 3.2: Relationship between Corr and Error on random vectors with 50 elements.

observe that the error between two vectors is inversely related with their correlation (see Figure 3.2). Therefore, by setting $\epsilon$ to the error of row $r \in R$ that has the smallest Corr$(M[r, C], t)$ above threshold $\rho_{\text{min}}$ (Line 2), it is possible to add columns to $C$ that do not reduce the fitness of the bicluster. This is possible because if the values in column $c$ are close to the extended tendency vector, adding column $c$ to the bicluster does not increase the error between any row and the tendency vector. Therefore, it also does not decrease the correlation of any row with the tendency vector.

After calculating $c$, the number of columns to add, CPB determines the actual number by multiplying $c$ by the so-called row-to-column ratio (actually this parameter is just the fraction of $c$ to include). The intent of this step is to allow CPB to find biclusters of varying dimensions, but it does not work as intended. For further discussion see Section 3.2.3.

As in the rows update step, the fixed row weight $\omega$ is used in calculating $t_{ext}$ and error values.
3.2 Improvements

3.2.1 Improving tendency vector calculation

Benchmarking CPB revealed that iteratively calculating the tendency vector contributed a significant amount of time to CPB’s running time. We were able to reduce that time by substituting a single-pass method for generating the tendency vector.

The original method required multiple iterations because the column mean of inversely correlated rows is not a good tendency vector. Figure 3.3(a) visualizes this problem: it shows the parallel coordinate plot for the rows of a 50 by 50 matrix. The absolute values of all pairwise correlations are 1.0, but half are inversely correlated. The column mean of this matrix is approximately the zero vector.

One simple solution, then, is to attempt to ensure that all rows of the bicluster are positively correlated. This can be done for bicluster matrix $X = M[R,C]$ by taking the additive inverse of each row that is negatively correlated with $X[0,*]$. 

Figure 3.3: Row expression levels before and after making all pairwise correlations positive.
Figure 3.4: Running time: iterative vs single-pass tendency vector calculation

Figure 3.3(b) demonstrates the result: clearly these rows are all positively correlated. Then the tendency vector may be calculated in one pass as the mean of the row vectors: \( t = \text{COLMEAN}(X_{mod}) \).

Generating the tendency vector in this way significantly speeds up CPB. Figure 3.4 compares the running time of CPB in seconds using these two methods as the number of rows in the dataset being clustered increases.

Though this method was only intended to improve CPB’s running time, it had the fortunate side effect of improving its accuracy, too. Figure 3.5 shows the relevance and recovery scores (explanation in Section 5.3.2) on a dataset with 500 rows, 200 columns, and one shift-pattern bicluster with 50 rows and 50 columns; CPB was run with 100 seeds and target \( r = 0.9 \). This improvement in score indicates that the
Figure 3.5: Scores: iterative vs single-pass tendency vector calculation

The single-pass tendency vector is a better estimate of the “true” tendency vector than is the iteratively-generated one.

These results suggest that the modified tendency vector works well in practice. However, it is not perfect. In particular, this method is not guaranteed to solve the problem of inversely correlated vectors canceling each other out, as illustrated in Figure 3.3. We offer a geometric explanation. The method works as follows: choose a vector \( \mathbf{w} \in \mathbb{R}^n \), which define a \( n - 1 \) dimensional hyperplane. Take the inverse of each vector “on the other side of that hyperplane from \( \mathbf{w} \)”, i.e., each vector with angle greater than \( \pi/2 \) with \( \mathbf{w} \). It is easy to see that any two vectors \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) that are both orthogonal to \( \mathbf{w} \) lie in the hyperplane and are unaffected by this process. Therefore, the angle between them is unconstrained: \( 0 \leq \theta_{\mathbf{v}_1, \mathbf{v}_2} \leq \pi \). So it is still possible for vectors to be negatively correlated, possible canceling each other out. However, although it is impossible to completely eliminate this problem, it is
theoretically possible to minimize it by choosing the correct hyperplane around which to invert the vectors. The problem then becomes:

$$\arg \min_{w \in \mathbb{R}^n} |\{v : v \in V \land w \perp v\}|$$

In words: find a vector $w$ that is orthogonal to as few vectors $v \in V$ as possible.

This is a possible avenue for further research. Preliminary empirical results suggest that one optimal way to compute the tendency vector is related to Principal Component Analysis [37]. After mean-shifting each vector, and normalizing each vector by its distance, the largest principal component, which may be easily computed by power iteration [34], appears to be a good tendency vector.

### 3.2.2 A lower bound for the minimum pairwise correlation

In Section 3.1.1, we discussed the merits of defining the fitness of a bicluster in terms of a tendency vector, rather than considering all pairwise row vector correlations. However, our ultimate goal is to ensure that all genes in the bicluster are correlated with each other. Given a bicluster’s PCC threshold $\rho_{\text{min}}$, which is the minimum allowed correlation between any gene and the tendency vector, we now show how to compute a lower bound on the minimum pairwise gene correlation, which we denote $\rho_{\text{pw}}$; this is just the bicluster’s fitness, as defined in Equation 3.4: $\rho_{\text{pw}} = \text{Fitness}(M[R, C])$.

Now we relate $\rho_{\text{min}}$ with $\rho_{\text{pw}}$ by using the geometry of the correlation coefficient. Without loss of generality, we assume that all row vectors in $M[R, C]$ are positively correlated; if they are negatively correlated, the following argument holds up to a change of sign.

To begin, note that a minimum correlation $\rho$ between two vectors implies a maximum angle $\theta = \arccos(\rho)$ between them. Therefore, a tendency vector $t$ and a
minimum correlation $\rho_{\text{min}}$ imply a cone within which all row vectors in the bicluster must lie. This cone is formed by all vectors $\mathbf{v}$ such that $\theta_{vt} \leq \theta_{\text{max}} = \arccos(\rho_{\text{min}})$; Figure 3.6 demonstrates the cone in the three-dimensional case.

Then maximum angle between any two vectors occurs when they each lie in the cone, so the maximum pairwise angle $\theta_{pw} = 2\theta_{\text{min}}$. The following manipulations then relate $\rho_{\text{min}}$ and $\rho_{pw}$:

\[
\theta_{\text{pw}} \leq 2\theta_{\text{min}} \\
\cos(\theta_{\text{pw}}) \geq \cos(2\theta_{\text{min}}) \\
\cos(\theta_{\text{pw}}) \geq 2\cos^2(\theta_{\text{min}}) - 1 \\
\rho_{\text{pw}} \geq 2\rho_{\text{min}}^2 - 1
\]

And we have a lower bound for $\rho_{pw}$. The inequality flips in the second step because $\cos$ is strictly decreasing on the interval $[0, \pi]$. 
Also, by solving for $\rho_{\text{min}}$, we get the extremely useful inverse relationship, which allows us to choose the appropriate $\rho_{\text{min}}$ for a given minimum fitness $\rho_{\text{pw}}$:

$$\rho_{\text{min}} \leq \sqrt{\frac{\rho_{\text{pw}} + 1}{2}}$$

We now know exactly how to choose $\rho_{\text{min}}$ to ensure that CPB finds biclusters with a known fitness: simple take the smallest possible upper bound for $\rho_{\text{min}}$, which occurs at $\sqrt{(\rho_{\text{pw}} + 1)/2}$. Figure 3.7 contains a plot of this function. Notice that when $\rho_{\text{pw}} = 0$, $\rho_{\text{min}} = \sqrt{1/2} \approx 0.707$. This means that when $\rho_{\text{min}} < \sqrt{1/2}$, there is no guarantee that genes in the resulting bicluster will be correlated at all: if $\rho_{\text{min}} = \sqrt{1/2}$, then $\cos(\theta_{\text{max}}) = \pi/2$. Then two vectors lying in the cone around $t$ may be orthogonal, so their correlation is 0. The plot also shows that when $\rho_{\text{min}} < \sqrt{1/2}$, $\rho_{\text{pw}}$ may be negative. As expected, in the extreme case when $\rho_{\text{pw}} = -1$, there is no restriction whatsoever, and even rows orthogonal to the tendency vector will be allowed in the bicluster.
3.2.3 Removing the row-to-column ratio

The original intent of the row-to-column ratio was to control the shape of the biclusters found by CPB. In practice, setting this parameter to any value other than 1.0 actually prevented CPB from discovering both asymmetrical and symmetrical biclusters.

Figure 3.8 demonstrates the effect of the row-to-column ratio on CPB’s recovery and relevance of both symmetrical and asymmetrical biclusters. The datasets each had 500 rows and 200 columns, each with one bicluster with varying dimensions. In all three cases (more rows, equal numbers of rows and columns, and more columns), using setting the parameter to 1 yielded the best recovery and relevance scores.

These results indicate that, by forcing CPB to include fewer columns (for values of the row to column ratio less than 1) or more columns (for values greater than 1)
than indicated by the error estimates, the row to column ratio is actually interfering with the algorithm’s optimal operation. Therefore we removed it.

### 3.2.4 Filtering biclusters found by chance

Datasets often contain small biclusters with high fitness by chance. CPB finds these biclusters, which typically have few rows and many columns, or vice versa. The number and size of such chance biclusters differs among datasets, so a simple nonparametric filtering method to identify and remove these biclusters would be useful. We have developed such a method, based on biclustering a shuffled version of the dataset.

Let $\mathbb{B} = \{B_1, B_2, \ldots, B_z\}$ be the set of biclusters found by CPB on a data matrix $M$. We generate a new dataset $M_s$ by shuffling the elements of $M$, and we run CPB, with the exact same parameters, on $M_s$ to generate a new set of biclusters $\mathbb{B}_s$. Then the size of largest bicluster (by area, i.e. the number of its elements, or the product of the number of rows and columns) in $\mathbb{B}_s$ is used as a threshold for filtering $\mathbb{B}$. Let $a$ be the area of the largest bicluster in $\mathbb{B}_s$; any biclusters with area equal to or smaller than $a$ is discarded from $\mathbb{B}$. The filtering algorithm is given in Algorithm 5.

#### Algorithm 5: Filter Random Biclusters.

**Data:** set of biclusters $\mathbb{B}$, data matrix $M$

**Result:**

1. begin
2. $M_s \leftarrow \text{SHUFFLE}(M)$;
3. $\mathbb{B}_s \leftarrow \{\}$;
4. for $i$ in 1 to $|\mathbb{B}|$ do
5.  $\mathbb{B}_s \leftarrow \mathbb{B}_s \cup \{\text{CPB}(M_s, \ldots)\}$;
6. end
7. $a_{\text{thresh}} \leftarrow \max(\{\text{AREA}(B) : B \in \mathbb{B}_s\})$;
8. return $\{B : B \in \mathbb{B}, \text{AREA}(B) > a_{\text{thresh}}\}$
9. end
Figure 3.9 shows the improvement in CPB’s relevance scores after filtering out biclusters in this way. This filtering method does improve the relevance of CPB’s results on synthetic data, but it still fails to totally remove all spurious biclusters. In addition, on gene expression data it removes many biclusters that are actually enriched with Gene Ontology terms. Given these limitations, there is likely room for further improvement.

3.3 Implementations

Because of the time required to iteratively compute the tendency vector, and because it is necessary to generate a large number of seeds to cover the entire dataset, CPB was originally implemented in C. That implementation is available at
http://bmi.osu.edu/hpc/software/cpb/. We have refactored that version and modified it to include my improvements. Because the modified version of CPB is so much more efficient at calculating the tendency vector, it became feasible to attempt a version in Python. Having a Python version available is useful for several reasons: the code base is smaller, the code is easier to understand, it is less likely to suffer from memory issues, and it is easier to modify.

By using NumPy array operations, which use BLAS and LAPACK, the Python version is actually only about two times slower than the unmodified C version (see Figure 3.10). Moreover, it could be further optimized by writing the inner loops in C. In the future we plan to combine the two implementations; the resulting code base will be significantly smaller, easier to understand, and easier to modify, without sacrificing speed.
Figure 3.10: Running times of original C version and Python version of CPB as the number of rows in the dataset increases.
CHAPTER 4

BIBENCH: A FRAMEWORK FOR BICLUSTERING ANALYSIS

Many biclustering algorithms have been published in the past decade. However, implementations are available for only a fraction of those algorithms. Those implementations that are available are written in different languages and with different interfaces. This fragmentation adds unnecessary steps to the task of biclustering gene expression data.

We have written a biclustering tool, BiBench, to help alleviate these difficulties. BiBench is a Python package that provides a common interface to many biclustering algorithms and can automatically download datasets from the Gene Expression Omnibus (GEO) [14]. BiBench also provides a toolbox of biclustering-related functionality, such as cluster evaluation and visualization, to streamline the whole biclustering process.

4.1 Examples of use

The following commands download GDS3715, find ten bicluster using the CPB algorithm, and perform Gene Ontology enrichment analysis on the genes of the first bicluster. By default, GO enrichment analysis uses the Biological Process Ontology and performs multiple test correction using the Benjamini & Hochberg method [23].
import bibench.all as bb

data = bb.get_gds_data('GDS3715')

found = bb.cpb(data, nclus=10)

bb.enrichment(found[0], data.annotation, data.genes)

### 4.2 Related work

Tools similar to BiBench already exist: biclust [27], BicAT [42], Expander, [45], BiGGEsTS [19], and BicOverlapper [43]. There are advantages and disadvantages to each tool, but BiBench provides features that are not available in the others, including more biclustering algorithms. BicAT, Expander, BiGGEsTS, and BicOverlapper are a graphical program, making them suitable for light exploratory work. However, for serious data mining tasks involving more than a few datasets or algorithms, automating the task with BiBench is easier. biclust is a library for R. BiBench provides bindings to all of biclust’s functionality, as well as extra biclustering algorithms not available in biclust. By using an interface to R, BiBench also has access to all of R’s other functionality. BicAT implements each algorithm, whereas BiBench uses the authors’ own implementations. Expander only provides one biclustering algorithm, SAMBA, though it does implement many other features for gene expression analysis and visualization. BiGGEsTS focuses specifically on biclustering time series data. BicOverlapper focuses on visualization. Like BiBench, it relies on R and Bioconductor for most of its functionality. BiBench can export data and biclusters for visualization in BicOverlapper.
4.3 Overview of functionality

4.3.1 Algorithms

BiBench currently supports the following biclustering algorithms: Bayesian Biclustering [21], BiMax [40], Cheng and Church [7], COALESCE [26], CPB [4], FABIA [24], ISA [3], OPSM [2], PLAID [50], QUBIC [32], Spectral [29], and xMOTIFs [35]. For a brief explanation of each algorithm, see Chapter 2; for CPB, see Chapter 3.

BiBench does not implement these algorithms itself; instead it provides bindings to the authors’ own implementations, which must be installed separately. However, the R packages biclust [27], isa2 [10], and fabia [24] provide most of the implementations, so they are easily installed. It is also easy to extend BiBench to support new algorithms.

4.3.2 Datasets

BiBench makes it convenient to download and cluster any Gene Expression Omnibus dataset and any dataset available as a package for Bioconductor [18]. Moreover, BiBench automatically infers the appropriate Bioconductor annotation package for each dataset, reducing the steps necessary to do enrichment analysis.

BiBench provides utilities for generating synthetic bicluster datasets from the following models: constant; constant upregulated; plaid, an additive model introduced by Lazzeroni and Owen [31]; shift-scale; shift; and scale. It also provides wrappers to the data creation methods of the fabia and isa packages.

Many algorithms require that data be transformed or normalized before clustering. BiBench provides functions for binarizing, discretizing, and normalizing datasets, and it provides a wrapper to the pcaMethods package for missing values imputation [47].
4.3.3 Validation

For comparing biclusters, BiBench adapts many similarity indices, such as the Jaccard coefficient and the F measure. To calculate the similarity of sets of biclusters, BiBench implements the set score of Prelić et al. [40]. BiBench also provides functions for Gene Ontology enrichment analysis with multiple test correction, using GOstats [15] and multtest [39].

4.3.4 Visualization

BiBench provides wrappers to biclust for the following visualizations: heatmap, bicluster projection, and parallel coordinates. It also allows exporting biclusters for visualization in BicOverlapper [43].

4.4 Availability

To obtain BiBench, download and install from http://bmi.osu.edu/hpc/software/bibench/.
(a) Heatmap. Shuffled so that three upregulated (red) and one downregulated (green) biclusters are contiguous.

(b) Parallel coordinates plot of the row and column vectors of a bicluster matrix.

(c) Bubble plot. Projection of biclusters onto two dimensions. This plot shows four large biclusters some small ones.

(d) BicOverlapper. Elements represent rows and columns. Hulls represent biclusters.

Figure 4.1: Four types of visualization supported by BiBench. The first three are performed by the biclust package; the last by BicOverlapper.
CHAPTER 5

COMPARISON OF BICLUSTERING ALGORITHMS

In this chapter we use BiBench to compare all twelve biclustering algorithms that it supports on both synthetic data and gene expression data.

Most biclustering problems are exponential in the rows and columns of the dataset, so algorithms must depend on heuristics, making their performance across the whole landscape of datasets suboptimal and difficult to evaluate. Moreover, because the ground truth of real biological datasets is unknown, it is difficult to verify a biclustering’s biological relevance. Because of these difficulties, there exists no consensus of which biclustering approaches are most promising. Moreover, attempts to reach such a consensus are soon outdated, because new algorithms are being published all the time.

In this chapter we further attempts at comparing biclustering algorithms by making the following improvements. First, we compare twelve biclustering algorithms, many of which have only recently been published and not extensively studied. Rather than using default parameters, each algorithm’s parameters were tuned for each dataset. Though each algorithm optimizes a different model, most similar publications generated synthetic datasets from only one bicluster model. In contrast, we use six different models to find the best for each algorithm. Finally, when comparing algorithms on gene expression data, previous papers used only one or two datasets,
often obtained from *S. cerevisiae* or *E. coli*. Most did not perform multiple test correction when performing Gene Ontology enrichment analysis. We used eight datasets from the Gene Expression Omnibus, all but one of which have over 12,000 genes, and biclusters were considered enriched only after multiple test correction.

### 5.1 Related work

Several systematic comparisons of biclustering methods have been published. Similar papers have also been published in statistics journals, comparing co-clustering methods.

Turner et al. [50] adapted the F-measure to biclustering and introduced a benchmark for evaluating biclustering algorithms.

Prelić et al. [40] compared several algorithms on both synthetic data with constant and constant-column biclusters and on real data. Synthetic data was used to test the effects of bicluster overlap and experimental noise; results were evaluated by defining a new scoring method, called *gene match score*, to compare biclusters’ rows; columns were not considered. For real data sets, results were compared using both Gene Ontology (GO) annotations and metabolic and protein-protein interaction networks.

Santamaría et al. [41] reviewed multiple validation indices, both internal and external, and adapted them to biclustering. de Castro et al. [11] evaluated biclustering methods in the context of collaborative filtering. Wiedenbeck and Krolak-Schwerdt [54] generalized the ADCLUS model Shepard and Arabie [46] and compared multiple algorithms in a Monte-Carlo study on data simulated from their model. Filippone et al. [17] adapted stability indices to evaluate fuzzy biclustering.

Bozdağ et al. [5] compared several algorithms with respect to their ability to detect biclusters with shifting and scaling patterns: rows in such biclusters are shifted, scaled versions of some base row vector. The effects of bicluster size, noise, and overlap
were compared on artificially-generated datasets. Results were evaluated by defining *external* and *uncovered* scores, which compare the area of overlap between the planted bicluster and found biclusters.

Chia and Karuturi [8] used a differential co-expression framework to compare algorithms on real microarray datasets.

## 5.2 Algorithms

The twelve algorithms compared in this chapter are those algorithms available in BiBench; see Section 4.3.1 and Table 5.1. Except for CPB, a summary of their operation appears in Chapter 2. The modified version of CPB was used in these experiments; the summary of CPB, including modifications, appears in Chapter 3.

The version of BBC used for these experiments was based on the publicly available code \(^1\), but we had to modify it to run on 64-bit machines and to fix a bug that caused it crash when seeking only a single bicluster.

### 5.2.1 Parameter selection

Choosing the correct parameters for each algorithm is crucial to that algorithm’s success, but too often default parameters are used when comparing algorithms. We chose parameters specifically for the synthetic and GDS data that worked better than the defaults.

For synthetic datasets, all algorithms that find a specific number of biclusters were given the true number of biclusters. Those that generate multiple seeds were given 300 seeds. For GDS datasets, those same algorithms were given 30 biclusters and 500 seeds, respectively, with two exceptions. Cheng and Church was given 100 biclusters, and

\(^1\)http://www.people.fas.harvard.edu/~junliu/BBC/
based on its author’s recommendations. BBC, which calculates the Bayesian Information Criterion (BIC) [44] for a clustering, was run multiple times on each GDS dataset, and the clustering with the best BIC was chosen. The number of clusters for each run was 30, 35, 40, 45, and 50.

BBC provides four normalization procedures, but no normalization worked best for constant biclusters. IQRN normalization worked best for the plaid model, so we chose to use IQRN on all tests, because BBC was designed to fit the plaid model.

Choosing the correct $\delta$ and $\alpha$ parameters are important for Cheng and Church’s accuracy and running time. $\delta$ controls the maximum mean-squared residue in the bicluster, and so affects the homogeneity of the results. On synthetic data were were able to get good results with $\delta = 0.1$, but on GDS data it needed to be increased. We used $\delta = e/2400$, where $e$ was the difference between the maximum and minimum values in the dataset. $\alpha$ controls the tradeoff between running time and accuracy; the minimum $\alpha = 1$ causes Cheng and Church to run as fast as possible. On synthetic data we were able to use $\alpha = 1.5$, but for the much larger GDS data it had to be reduced to 1.2.

The accuracy of BiMax, xMOTIFs, and QUBIC depends on how the data is discretized. xMOTIFs performed best on synthetic data discretized to a large number of levels; we used 50. As the levels decreased, so xMOTIFs’ performance suffered. For BiMax, which requires binary data, we used the discretization method used for QUBIC with two levels. QUBIC also performed best on synthetic datasets with only two levels. On GDS data, QUBIC got better results with the default of ten ranks.

The Spectral biclustering algorithm performed poorly on synthetic data until we reduced the number of eigenvalues to one, used bistochastization normalization, and increased the within-bicluster variance to five. On GDS data, it got better results using log normalization and a much larger variance. It failed to return any biclusters.
until the within-bicluster variance was extremely large, so we set it to twice the number of rows in the dataset.

5.3 Synthetic dataset experiments

Comparing algorithms by using synthetically-generated data cannot replace using real data, but synthetic datasets complement real ones for two reasons. First, the data may be manipulated to create specific experiments that test a particular property of the algorithms, such as their ability to resist varying levels of error. Second, the ground truth is known, allowing an accurate evaluation of each algorithm’s performance.

The algorithms were tested on a suite of synthetically-generated datasets, such that the set of true biclusters in each dataset is known. We first describe the data generation model and the similarity measure used to evaluate results, then give the results of testing the algorithms on their ability to recover biclusters under the following conditions: varying the bicluster model, additive noise, varying number of biclusters, and overlapping biclusters.

5.3.1 Synthetic data model

In all of the following experiments, synthetic datasets were generated with the following parameters, except when one parameter was varied in an experiment: 500 rows, 200 columns, one biclusters with 50 rows and 50, no noise, no overlap. Datasets from six different models of biclustering were generated:

- **Constant biclusters**: Biclusters with a constant expression level close to dataset mean. The constant expression values of the biclusters were chosen to be 0; background values were independent, identically-distributed (i.i.d.) draws from the standard normal: $N(0,1)$. 
• *Constant, upregulated biclusters*: Like the previous model, but biclusters had a constant expression level of 5.

• *Shift-scale biclusters*: Each bicluster row is both shifted and scaled from some base row. Base row, shift and scale parameters, and background were all i.i.d. \( \sim N(0, 1) \).

• *Shift biclusters*: Like shift-scale, but scaling parameters always equal 1.

• *Scale biclusters*: Like shift-scale, but shifting parameters always equal 0.

• *Plaid model biclusters*: An additive bicluster model first introduced by Lazzeroni and Owen [31]. Each element \( X_{ij} \), with \( K \) biclusters assumed present, is modeled as the sum of a background effect \( \theta \), cluster effects \( \mu \), row effects \( \alpha \), column effects \( \beta \), and random noise \( e \). \( \rho \) and \( \kappa \) are indicator variables for row \( i \) and column \( j \) membership in bicluster \( k \):

\[
X_{ij} = \theta + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk})\rho_{ik}\kappa_{jk} + e_{ij}
\]

All effects were chosen i.i.d. \( \sim N(0, 1) \).

It is important to note that, with the exception of the plaid model, these models are meant only for testing, not modeling data. They were chosen because they are representative of most of the types of biclusters found by the twelve algorithms under investigation. They were also easy to generate, because most are special cases of shift-scale biclusters. Table 5.1 gives, for each algorithm, the models it should theoretically be able to recover.

### 5.3.2 Bicluster similarity measures

There have been many approaches to calculating the similarity of two biclusters. For this thesis the similarity \( s(b_1, b_2) \) of two biclusters \( b_1, b_2 \) was chosen to be the Jaccard
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>const</th>
<th>const-upreg</th>
<th>plaid</th>
<th>scale</th>
<th>shift</th>
<th>shift-scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BiMax</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cheng and Church</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COALESCE</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>FABIA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPSM</td>
<td></td>
<td>✔</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plaid</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUBIC</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xMOTIFs</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Theoretical support of each bicluster model by algorithm.

Coefficient applied to submatrix elements defined by each bicluster. If $b_1 = (R_1, C_1)$ and $b_2 = (R_2, C_2)$, the Jaccard coefficient is calculated as:

$$J_{\text{ACC}}(b_1, b_2) = \frac{|b_1 \cap b_2|}{|b_1 \cup b_2|}$$

(5.1)

where $|b_1 \cap b_2|$ is the number of data elements in their intersection, and $|b_1 \cup b_2|$ is the number in their union (not the area of their union). Figure 5.1 visualizes the intersection and union of two biclusters; the Jaccard coefficient is the area of the dark rectangle formed by their intersection divided by the total area of the shaded regions. Formally, it is calculated as follows:

$$J_{\text{ACC}}(b_1, b_2) = \frac{|R_1 \cap R_2||C_1 \cap C_2|}{|R_1||C_1| + |R_2||C_2| - |R_1 \cap R_2||C_1 \cap C_2|}$$

(5.2)

Identical biclusters achieve the largest score of $s(b_1, b_2) = 1$, and disjoint biclusters the lowest of $s(b_1, b_2) = 0$. Any score $x \in [0, 1]$ is easily interpreted as the percentage $x$ of total elements shared by both biclusters.

An similarity score for sets of biclusters was calculated with the following method, adapted from Prelić et al. [40]. Let $b_1$ and $b_2$ be biclusters. Let $s(b_1, b_2)$ be some score
Figure 5.1: The intersection area (dark red) and union area (all shaded areas) involved in the calculation of the Jaccard index.

function which compares biclusters; without loss of generality assume that $s$ assigns scores to similar biclusters and small scores to dissimilar ones. Then two sets of biclusters, $M_1$ and $M_2$, are compared by calculating the set score $S(M_1, M_2)$:

$$S(M_1, M_2) = \frac{1}{|M_1|} \sum_{b_1 \in M_1} \max_{b_2 \in M_2} s(b_1, b_2)$$

(5.3)

Because $S$ is not symmetric, it is used to define two scores, recovery and relevance, depending on the order of the expected and found biclusters. Let $E$ denote the ground truth set of expected biclusters, and $F$ denote the set of found biclusters. Recovery is calculated as $S(E, F)$; it is maximized if $E \subseteq F$; i.e., if the algorithm found all of the expected biclusters. Similarly, relevance is calculated as $S(F, E)$; it is maximized if $F \subseteq E$, i.e., all the found biclusters were expected. If $S(E, F) = S(F, E) = 1$, then $E \subseteq F$ and $F \subseteq E$, so $E = F$.

5.3.3 Model experiment

In many previous papers, algorithms were compared on artificial data generated from a single model. However, each algorithm fits a different bicluster model. To compare algorithms on a single model, then, often gives incomplete or misleading results.
Instead, we evaluated each bicluster on synthetic datasets generated from six different models: constant, constant-upregulated, shift, scale, shift-scale, and plaid.

Twenty datasets for each of the six models were generated, and each biclustering method was scored on each dataset. Plots of the mean recovery and relevance scores for all twenty datasets of each model are given in Figure 5.2.

BiMax, ISA, and OPSM do not filter their results, so they each return many spurious biclusters that hurt their relevance scores. However, BiBench provides a bicluster filtering function that removes biclusters based on size and overlap with other biclusters. After filtering with 25% maximum overlap, their recovery scores (Figure 5.3) were much improved.

All algorithms were not expected to perform well on all datasets; see Table 5.1. Most algorithms were able to recover biclusters that fit their model, but there were a few exceptions.

BBC’s results are sensitive to which normalization procedure it uses. Depending on the procedure chosen, it is capable of achieving perfect scores on constant, constant-upregulated, shift, and plaid biclusters. We chose to use IQRN normalization, which maximized its performance on plaid-model biclusters.

Cheng and Church was expected to find any biclusters with constant expression value, but it could not find upregulated constant biclusters. We hypothesize that rows and columns with large expression values were pruned early because they increased the mean-square residue of the candidate bicluster.

Since all the biclusters except constant and constant-upregulated were instances of order-preserving submatrices, OPSM was expected to succeed on these datasets. However, it did not perform well on scale or shift-scale biclusters. These failures are due to OPSM’s method of scoring partial biclusters: it awards high scores for large gaps between expression levels, so biclusters with small or nonexistent gaps get
Figure 5.2: Bicluster model experiment. Each data point represents the mean of twenty datasets. A score of (1,1) is best.
pruned early in the search process. In these datasets, scale and shift-scale biclusters had small gaps because the scaling factors for each row were drawn from a standard normal distribution, contracting most rows towards zero and thus shrinking the gap statistic.

CPB was expected to do well on both constant and upregulated bicluster models. It cannot perfectly recover constant biclusters, because the correlation of two constant vectors is undefined, but it should recover the biclusters plus a few extra columns. It was able to recover constant biclusters, but as the bicluster upregulation increased, the CPB’s recovery decreased. This behavior makes sense because CPB finds biclusters with high row-wise correlation. Increasing the bicluster upregulation also increases the correlation between any two rows of the data matrix that contain upregulated portions. Generating more bicluster seeds allowed CPB to recover the upregulated biclusters.

FABIA only performed well on constant-upregulated biclusters, but it is important to note that it is capable of finding other bicluster models not represented in
this experiment. The parameters for these datasets were generated from Gaussian
distributions, whereas FABIA is optimized to perform well on data generated from
distributions with heavy tails.

Some algorithms also performed unexpectedly well on certain data models. CO-
ALESCE, ISA, and QUBIC were able to partially recover plaid-model biclusters by
recovering the upregulated portions. BBC was able to partially recover shift-scale
patterns.

In subsequent experiments, each algorithm was tested on datasets generated from
the biclustering model on which it performed best in this experiment. Most did best
on constant-upregulated biclusters. CPB and OPSM did best on shift biclusters,
BBC on plaid-model biclusters, and Cheng and Church on constant biclusters.

5.3.4 Noise experiment

Data is often perturbed both by noise inherent in the system under measurement
and by errors in the measuring process. The errors introduced from these sources
lead to noisy data, in which some or all of the signal has been lost. Algorithms
robust with respect to noise are preferable for any data analysis task. Therefore,
the biclustering algorithms were compared on their ability to resist random noise in
the data. Each dataset was perturbed by adding noise generated from a Gaussian
distribution with zero mean and a varying standard deviation $\epsilon$: $N(0, \epsilon)$. To allow
fair comparisons between algorithms, they were run on all four datasets on which at
least one algorithm performed best in the previous experiment. To make the higher
noise levels more meaningful, the constant upregulated biclusters were generated with
a bicluster mean of three, instead of five. The results for noise experiment appear in
Figures 5.4.

As expected, increasing the random noise in the dataset negatively affected both
Figure 5.4: Noise experiments: additive noise. Dots shows the mean score; lines show one standard deviation.
the recovery and relevance of clustering returned by most algorithms. BBC, COALESCE, ISA, and Plaid were most resistant to noise on constant upregulated bicluster datasets. COALESCE had the best performance: even when the error standard deviation equalled the upregulation factor, it perfectly recovered the biclusters.

In general the algorithms which seek local patterns, such as BiMax, Cheng and Church, CPB, OPSM, and xMOTIFs, were more sensitive to noise, whereas the algorithms that fit a model of the entire dataset, such as ISA, FABIA, COALESCE, and Plaid were less sensitive. We hypothesize that modeling the entire dataset makes most algorithms more robust because it uses all the available information in the data. There were exceptions to this pattern, however. QUBIC handled noise much better than did other algorithms that seek local patterns; in contrast, though we used QUBIC’s method for binarizing the dataset for BiMax, its scores were more sensitive to noise. This result makes sense because BiMax’s biclusters are fragile: introducing just a few zeros to a bicluster is likely to noticeably reduce its recovery.

OPSM is especially sensitive to noise because even relatively small perturbations may affect the ordering of rows.

We hypothesized that xMOTIFs’s poor performance was due to the large number of levels used when discretizing the data, but reducing the number of levels did not improve its score.

5.3.5 Number experiment

Most gene expression datasets are not likely to have only one bicluster; large datasets with hundreds of samples and tens of thousands of probes may contain hundreds or thousands of biclusters. Therefore in this experiment the algorithms were tested on their ability to find increasing numbers of biclusters. The datasets in this experiment
Figure 5.5: Number experiment: increasing number of biclusters.
had 250 columns; the number of biclusters varied from 1 to 5. The results appear in Figure 5.5.

BBC, COALESCE, CPB, ISA, QUBIC, and xMOTIFs were unaffected by the number of biclusters. In fact, CPB’s, ISA’s, and xMOTIFs’s relevance scores actually improved as the number of biclusters in the dataset increased.

Even when the number of biclusters is known, recovering them accurately can be challenging, as evidenced by the trouble the other algorithms had as the number increased. Plaid and OPSM were most affected, whereas the degradation in the other algorithms’ performances was more gradual.

These scores were calculated with the raw results; after filtering as described in Section 5.3.3, ISA’s recovery and relevance scores dropped to around 0.25 once more than one bicluster was present. This behavior was caused by ISA finding a large bicluster that was a superset of all the planted biclusters.

5.3.6 Overlap experiment

Algorithms were also tested on their ability to recover overlapping biclusters. The overlap datasets were generated with 2 embedded biclusters, each with 50 rows and 50 columns. In each dataset, bicluster rows and columns overlapped by 0, 10, 20, and 30 elements. Past this point, the biclusters become increasingly indistinguishable, and so the recovery and relevance scores approach those for datasets with one bicluster.

The bicluster expression values in overlapping regions were not additive, with the exception of the plaid model. Shift biclusters were generated by choosing the shift and scale parameters in a way to let two biclusters have same expression values at overlapping areas. The results for overlapping columns and rows appear in Figure 5.6.

A few algorithms were relatively unaffected by overlap. ISA’s scores did not
Figure 5.6: Overlap experiment: increasing amounts of overlap between two clusters.
change, Plaid’s scores actually improve until the overlap degree reaches thirty. CPB’s relevance score dropped slightly, but it was otherwise unaffected.

OPSM’s recovery scores increased, but only because its initial score was low, suggesting that it could only find one bicluster. As the bicluster it did not find overlapped with the one that it did, the overlap area boosted the recovery score.

Most other algorithms’ scores were negatively affected by overlapping the biclusters. In particular, Spectral’s scores plummeted; most other algorithms’ scores decreased gradually. BBC’s drop in score was expected, because it actually fits a modified plaid model that does not allow overlapping biclusters.

5.4 Microarray dataset experiments

Microarrays enables the collection of vast amounts of gene expression data from biological systems. A single microarray chip can collect expression levels from thousands of genes, and this data is often collected from multiple tissues, in multiple patients, with different medical conditions, at different times, and in multiple trials. For instance, the Gene Expression Omnibus (GEO), a public database of gene expression data, currently contains 659,203 samples on 9,528 different microarray platforms [14]. The availability of these high-dimension datasets makes them a good target for data mining techniques such as biclustering.

The Gene Expression Omnibus is organized into the following categories:

- Platform (GPL): Information about the technology, such as microarray or sequencer, used to generate the data.

- Sample (GSM): Conditions and measurement from a single sample.

- Series (GSE): A set of related samples.
Datasets (GDS): A curated series, with all samples from the same platform.

The Gene Expression Omnibus currently contains 9,562 platforms, 665,126 samples, 26,848 series, and 2,720 datasets.

For biclustering, we are interested in GDS datasets. The \( m \times n \) expression matrix derived from a GDS dataset contains \( n \) samples, each with a length-\( m \) gene expression profile. Each of the \( m \) rows represents a probe for a particular gene.

Most GDS datasets used in this work had missing values. These missing values were replaced using PCA imputation, provided by the pcaMethods package [47].

5.4.1 Gene Ontology enrichment analysis

A different method must be used for evaluating the results of biclustering gene expression data, because the true biclusters are not known. Two classes of methods are available: internal and external. Internal measures of validity are based on intrinsic properties of the data and biclusters themselves; external measures compare the biclusters to some other source of information. Because we compared so many algorithms, each fitting a different model, we chose to use external validation by calculating Gene Ontology enrichment for the rows of each bicluster.

The Gene Ontology project consists of three vocabularies for bioinformatics: cellular component, molecular function, and biological process. Each ontology is a tree of terms, so that terms have a parent-child relationship; child terms are more specific than any of their parents.

The genomes of many organisms have been annotated with Gene Ontology terms, depending on the role and behavior of their gene products. Given a set of genes chosen from a so-called 'gene universe', we may carry out a hypergeometric test to determine if a particular term associated with those genes occur more than predicted by random chance. When testing multiple terms, multiple test correction should be
used. Moreover, because terms have a parent-child relationship, a gene annotated with term also inherits all of that term’s descendants. The full enrichment analysis needs to account for these relationships between terms.

In this work the enrichment analysis was carried out using the GOstats package [15], which deals with term inheritance by conditioning on all significant descendant terms. Terms were chosen from the Biological Process Ontology. The genes associated with each probe in the bicluster were used as the test set; the genes associated with all the probes of each GDS dataset were chosen as the gene universe. Multiple test correction was performed using the Benjamini and Hochberg method [23]. Biclusters were considered enriched if the adjusted \( p \)-value for any gene ontology term was smaller than \( p = 0.05 \).

### 5.4.2 GDS results

Algorithms were compared on eight gene expression datasets from the Gene Expression Omnibus (GEO) database: GDS181 [48], GDS589 [52], GDS1027 [13, 57], GDS1319 [1], GDS1406 [25], GDS1490 [58], GDS3715 [55], and GDS3716 [20]. The datasets are summarized in Table 5.2.

The number of biclusters found and enriched for each algorithm appear in Table 61.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Genes</th>
<th>Samples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDS181</td>
<td>12559</td>
<td>84</td>
<td>Human &amp; mouse</td>
</tr>
<tr>
<td>GDS589</td>
<td>8799</td>
<td>122</td>
<td>Rat peripheral &amp; brain regions</td>
</tr>
<tr>
<td>GDS1027</td>
<td>15866</td>
<td>154</td>
<td>Rat lung SM exposure model</td>
</tr>
<tr>
<td>GDS1319</td>
<td>22548</td>
<td>123</td>
<td>C blastomere mutant embryos</td>
</tr>
<tr>
<td>GDS1406</td>
<td>12422</td>
<td>87</td>
<td>Mouse brain regions</td>
</tr>
<tr>
<td>GDS1490</td>
<td>12422</td>
<td>150</td>
<td>Mouse neural &amp; body tissue</td>
</tr>
<tr>
<td>GDS3715</td>
<td>12559</td>
<td>110</td>
<td>Human skeletal muscles</td>
</tr>
<tr>
<td>GDS3716</td>
<td>22215</td>
<td>42</td>
<td>Breast epithelia: cancer patients</td>
</tr>
</tbody>
</table>

Table 5.2: GDS datasets
Figure 5.7: Number of rows in all GDS biclusters.
Figure 5.8: Number of rows in enriched GDS biclusters.
Figure 5.9: Number of rows in enriched, non-overlapping GDS biclusters.
5.3. Biclusters were considered enriched if at least one term from the Biological Process Gene Ontology was enriched at the $p = 0.05$ level after Benjamini and Hochberg multiple test correction [23]. The last column in Table 5.3 shows the number of enriched biclusters after filtering out biclusters that overlapped. For instance, none of BBC’s enriched biclusters overlapped, but only nine of BiMax’s were sufficiently different. CPB found the most enriched biclusters, both before and after filtering. Moreover, CPB found biclusters with relatively few rows: Figures 5.7, 5.8, and 5.9. GO enrichment significance is correlated with the number of rows; this strengthens the argument that CPB is finding biologically relevant biclusters, not extremely large biclusters with inflated significance scores.

Though some algorithms found more enriched biclusters than others, further work is required to fully explore those biclusters and ascertain their biological relevance. It is important to note that COALESCE was designed to use genetic sequence data

<table>
<thead>
<tr>
<th>algorithm</th>
<th>found</th>
<th>enriched</th>
<th>filtered</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC</td>
<td>285</td>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td>BiMax</td>
<td>654</td>
<td>165</td>
<td>9</td>
</tr>
<tr>
<td>Cheng and Church</td>
<td>800</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>COALESCE</td>
<td>574</td>
<td>269</td>
<td>8</td>
</tr>
<tr>
<td>CPB</td>
<td>1312</td>
<td>463</td>
<td>347</td>
</tr>
<tr>
<td>FABIA</td>
<td>240</td>
<td>72</td>
<td>6</td>
</tr>
<tr>
<td>ISA</td>
<td>140</td>
<td>69</td>
<td>9</td>
</tr>
<tr>
<td>OPSM</td>
<td>140</td>
<td>52</td>
<td>12</td>
</tr>
<tr>
<td>Plaid</td>
<td>40</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>QUBIC</td>
<td>108</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>Spectral</td>
<td>769</td>
<td>381</td>
<td>65</td>
</tr>
<tr>
<td>xMOTIFs</td>
<td>153</td>
<td>34</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 5.3: Aggregated results on all eight GDS datasets. Biclusters were considered enriched if any GO term was enriched with $p = 0.05$ level after multiple test correction. The set of enriched biclusters was further processed to discard overlapping biclusters.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>rows, cols</th>
<th>terms (p-value)</th>
</tr>
</thead>
</table>
| BBC       | 94, 117    | translational elongation (2.00e-30)  
|           |            | cellular biosynthetic process (1.38e-06)  
|           |            | glycolysis (7.37e-06)  
|           |            | hexose catabolic process (3.64e-05)  
|           |            | macromolecule biosynthetic process (1.20e-04)  |
| BiMax     | 42, 9      | chromatin assembly or disassembly (2.75e-02)  |
| Chng&Chrh | 539, 91    | epithelial tube morphogenesis (9.94e-04)  
|           |            | branching inv. in ureteric bud morphogenesis (4.26e-02)  
|           |            | morphogenesis of a branching structure (4.26e-02)  
|           |            | organ morphogenesis (4.26e-02)  
|           |            | response to bacterium (4.26e-02)  |
| COALESCE  | 103, 122   | translational elongation (6.75e-12)  
|           |            | glycolysis (2.88e-03)  
|           |            | energy derivation by ox. of organic cmpnds (5.57e-03)  
|           |            | hexose catabolic process (5.57e-03)  
|           |            | ATP synthesis coupled electron transport (1.47e-02)  |
| CPB       | 229, 98    | oxoacid metabolic process (2.83e-13)  
|           |            | oxidation-reduction process (2.72e-08)  
|           |            | cellular amino acid metabolic process (4.82e-04)  
|           |            | monocarboxylic acid metabolic process (2.63e-03)  
|           |            | L-phenylalanine catabolic process (1.30e-02)  |
| FABIA     | 56, 28     | translational elongation (3.22e-17)  
|           |            | macromolecule biosynthetic process (2.99e-06)  
|           |            | protein metabolic process (4.12e-05)  
|           |            | translation (4.12e-05)  
|           |            | cellular macromolecule metabolic process (2.12e-04)  |
| ISA       | 292, 11    | translational elongation (1.44e-05)  
|           |            | protein metabolic process (5.35e-12)  
|           |            | RNA processing (2.26e-09)  
|           |            | biosynthetic process (4.19e-09)  
|           |            | rRNA processing (1.47e-08)  |
| OPSM      | 378, 11    | multicellular organism reproduction (2.78e-04)  
|           |            | gamete generation (1.31e-03)  
|           |            | neg. regulation of programmed cell death (2.92e-03)  
|           |            | spermatogenesis (6.90e-03)  
|           |            | anti-apoptosis (4.31e-02)  |
| Plaid     | 22, 15     | translational elongation (6.29e-30)  
|           |            | macromolecule biosynthetic process (1.78e-10)  
|           |            | protein metabolic process (3.13e-09)  
|           |            | cellular biosynthetic process (9.09e-08)  
|           |            | cellular macromolecule metabolic process (1.60e-06)  |
| QUBIC     | 40, 8      | gamete generation (1.95e-02)  
|           |            | death (1.99e-02)  
|           |            | regulation of cell death (3.55e-02)  
|           |            | neg. rgltn. DNA damage response ... p53 ... (4.64e-02)  
|           |            | neg. rgltn of programmed cell death (4.64e-02)  |
| Spectral  | 192, 73    | glycolysis (1.08e-05)  
|           |            | organic acid metabolic process (1.08e-05)  
|           |            | glucose metabolic process (4.51e-05)  
|           |            | hexose catabolic process (4.51e-05)  
|           |            | monosaccharide metabolic process (6.89e-05)  |
| xMOTIFs   | 50, 7      | translational elongation (7.89e-12)  
|           |            | ncRNA metabolic process (2.76e-03)  
|           |            | rRNA processing (3.51e-03)  
|           |            | cellular protein metabolic process (1.23e-02)  
|           |            | anaphase-promoting ... catabolic process (2.63e-02)  |

Table 5.4: Five most enriched terms for each algorithm’s best bicluster on GDS589.
in conjunction with gene expression data, but sequence data was not used in this test; if it had, COALESCE’s performance may have differed.

A full analysis of all the biclusters is outside the scope of this work, but we did examine best biclusters found by each algorithm. All twelve algorithms found enriched biclusters in GDS589. The terms associated with the bicluster with the lowest \( p \)-value for each algorithm appear in Table 5.4. The results are suggestive, considering that GDS589 represents gene expression of brain tissue. Most biclusters were enriched with terms related to protein biosynthesis. CPB’s bicluster contained proteins involved with the catabolism of L-phenylalanine, an essential amino acid linked with brain development disorders in patients with phenylketonuria [38]. OPSM found a bicluster with almost 400 genes enriched with anti-apoptosis and negative regulation of cell death terms, which are important for neural development [53]. Similarly, QUBIC’s bicluster was enriched with terms involving cell death and gamete generation. xMOTIFs and ISA both found biclusters enriched with RNA processing terms. BBC, COALESCE, and Spectral all found biclusters enriched with glycolysis, glucose metabolism, and hexose catabolism; these are interesting especially because mammals’ brains typically use glucose as their main source of energy [28].

5.5 Discussion

This chapter compared twelve biclustering algorithms on both synthetic data and gene expression data. The following patterns emerged:

- Choosing the correct parameters for each algorithm was crucial. Many similar publications used default parameters, which often yielded poor results in this study. Some algorithms, like Cheng and Church, may also exhibit excessive running time if parameters are not chosen carefully.
• As expected, each algorithm performed best on different biclustering models. Before concluding that one algorithm outperforms another, it is important to consider the kind of data on which they were compared.

• Algorithms that model the entire dataset seem more resilient to noise than algorithms that seek individual biclusters.

• The performance of most algorithms tested in this chapter degraded as the number of biclusters in the dataset increased. This is especially a concern for large gene expression datasets, which may contain hundreds of biclusters.

• No algorithm was able to fully separate biclusters with substantial overlap. CPB’s performance was least affected by overlapping biclusters, which caused a slight decrease in its relevance scores.

• In gene expression data, all algorithms were able to find biclusters enriched with GO terms. CPB found the most, followed by BBC. Surprisingly, the oldest of the biclustering algorithms, Cheng and Church, found the third most number of enriched biclusters.

• Performance on synthetic datasets did not always correlate with performance on gene expression datasets. For instance, the Spectral algorithm was highly sensitive to noise, number of biclusters, and overlap in synthetic data, but was able to find many enriched biclusters in gene expression data.
CHAPTER 6
CONCLUSION AND FUTURE WORK

In this thesis we described algorithmic improvements to the Correlated Patterns Bi-clustering algorithm that improve its running time, accuracy, and sensitivity to noise. We introduced BiBench, an integrated environment for bicluster analysis written in Python. Finally, we used BiBench to compare twelve biclustering algorithms on both synthetic data and gene expression data. In these comparisons CPB consistently performed better than most other algorithms overall: it was capable of discovering all but two bicluster models, except for noise it was unaffected in the synthetic data experiments, and in GDS data it recovered the most biclusters enriched with gene ontology terms. BiBench was invaluable in writing these experiments: by taking care of the details such as interfacing with algorithm implementations and downloading datasets, it freed us to focus on the experiments themselves.

There is still work to be done to improve the CPB algorithm, as suggested in Chapter 3. When calculating the tendency vector, the problem of choosing the optimal hyperplane for inverting the row vectors of a bicluster remains open. Further, our method of filtering spurious biclusters was not strict enough on synthetic datasets, but it was too strict on GDS datasets.

Biclustering has been successfully applied to bioinformatics and text mining, but
there are many other fields generating large datasets that could be explored with biclustering methods. The specific requirements of these fields will likely drive innovation of new biclustering techniques. For instance, neuroscience research is generating datasets that are orders of magnitude larger than gene expression datasets; many existing biclustering methods will not easily scale up to such large datasets. To deal with such large datasets, parallelization strategies such as implementing algorithms on GPU architectures will likely be necessary. It may be possible to adapt to CPB to work on neuroscience data. On even a small fMRI-sized dataset (100,000 voxels, 80 samples) CPB required about thirty seconds to recover a single bicluster (running on an Intel Core 2 Quad). However, the task of finding multiple biclusters with CPB is embarrassingly parallel, so it should be possible to cluster such a dataset in a reasonable amount of time by using a computing cluster. Whether CPB is capable of recovering relevant biclusters in such large datasets remains to be seen, but it is a potentially fruitful area of research.

To help researchers choose from the variety of existing biclustering methods, there is a need for more theoretical comparisons and empirical testing. This thesis and many others have attempted to contribute, but the job is too large for any one publication or any one team.

Finally, as biclustering’s popularity continues to grow, the demand for tools will also grow. As discussed in Chapter 4, there are existing software packages for bicluster analysis, including our own BiBench software. However, only a fraction of published algorithms have been implemented, and very few machine learning and statistical libraries currently include biclustering algorithms. We encourage any researchers developing a biclustering algorithm to make its source code freely available.
BIBLIOGRAPHY


72


