OXFORD
UNIVERSITY PRESS
Bioinformatics

EBIC: an evolutionary-based parallel biclustering algorithm for pattern discovery

Journal:	*Bioinformatics*
Manuscript ID	BIOINF-2018-0175.R1
Category:	Original Paper
Date Submitted by the Author:	07-Apr-2018
Complete List of Authors:	Orzechowski, Patryk Sipper, Moshe; University of Pennsylvania Perelman School of Medicine Huang, Xiuzhen; Arkansas State University, Computer Science Moore, Jason; University of Pennsylvania, Genetics
Keywords:	Machine learning, Cluster analysis, Genomics

Data and text mining

EBIC: an evolutionary-based parallel biclustering algorithm for pattern discovery

Patryk Orzechowski [1,2]*, Moshe Sipper [3], Xiuzhen Huang [4], and Jason H. Moore [1]*

[1] Institute for Biomedical Informatics, University of Pennsylvania, Philadelphia, PA 19104, USA, and
[2] Department of Automatics and Biomedical Engineering, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland, and
[3] Department of Computer Science, Ben-Gurion University, Beer Sheva 8410501, Israel,and
[4] Department of Computer Science, Arkansas State University, Jonesboro, AR 72467, USA

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Biclustering algorithms are commonly used for gene expression data analysis. However, accurate identification of meaningful structures is very challenging and state-of-the-art methods are incapable of discovering with high accuracy different patterns of high biological relevance.
Results: In this paper a novel biclustering algorithm based on evolutionary computation, a subfield of artificial intelligence (AI), is introduced. The method called EBIC aims to detect order-preserving patterns in complex data. EBIC is capable of discovering multiple complex patterns with unprecedented accuracy in real gene expression datasets. It is also one of the very few biclustering methods designed for parallel environments with multiple graphics processing units (GPUs). We demonstrate that EBIC greatly outperforms state-of-the-art biclustering methods, in terms of recovery and relevance, on both synthetic and genetic datasets. EBIC also yields results over 12 times faster than the most accurate reference algorithms.
Availability: EBIC source code is available on GitHub at `https://github.com/EpistasisLab/ebic`
Contact: Correspondence and requests for materials should be addressed to P.O. (email: patryk.orzechowski@gmail.com) and J.H.M. (email: jhmoore@upenn.edu)
Supplementary information: Supplementary Data with results of analyses and additional information on the method is available at *Bioinformatics* online.

1 Introduction

Discovering meaningful patterns in complex and noisy data, especially biological one, is a challenge. Traditional clustering approaches such as k-means or hierarchical clustering are expected to group similar objects together and to separate dissimilar objects into distinctive groups. These methods assume that all object features contribute to the classification result, which renders clustering a valuable technique for global similarity detection. Clustering does not, however, succeed when only some subset of features is important to a specific cluster.

The inability to capture local patterns is one of the main reasons for the advent of biclustering techniques, where biclusters – subsets of rows and columns – are sought. Both rows and columns subsets may contain elements that are not necessarily adjacent to each other, thus differentiating biclustering from other problems of pattern matching (e.g., image recognition), making also the task unsuitable for deep learning (Ching *et al.*, 2017).

Biclustering has its roots in data partitioning into subgroups of approximately constant values (Morgan and Sonquist, 1963) and simultaneously clustering rows and columns of a matrix (Hartigan, 1972); this was later called biclustering (Mirkin, 1996). For the last two decades biclustering has been applied to multiple domains, including biomedicine, genomics (especially gene expression analysis), text-mining, marketing, dimensionality reduction, and others (Busygin *et al.*, 2008; Dolnicar *et al.*, 2012).

Designing biclustering algorithms involves many challenges. First, although over fifty biclustering algorithms have been proposed (much more when derivatives are considered), no method has proven capable of detecting – with sufficient accuracy – six major types of patterns that are commonly present in gene expression data. Most biclustering algorithms find only one or a few of these patterns (Madeira and Oliveira, 2004; Eren *et al.*, 2013; Pontes *et al.*, 2015a; Wang *et al.*, 2016; Padilha and Campello, 2017): column-constant, row-constant, shift (i.e., additive coherent), scale (i.e., multiplicative coherent), shift and scale (i.e., simultaneous coherent), and order-preserving. Detection of order-preserving patterns is especially important, because it may be considered a generalization of the five other patterns (Ben-Dor *et al.*, 2003; Eren *et al.*, 2013; Wang *et al.*, 2016).

Second, many biclustering algorithms are unable to detect negative correlations or capture approximate patterns. Moreover, biclustering algorithms fail to properly separate partially overlapping biclusters. The performance of these algorithms on overlapping problems usually drops dramatically with increasing levels of overlap (Wang *et al.*, 2016).

A third drawback of current biclustering methods is their limited success assessing which biclusters are the most relevant. Multiple measures for assessing quality of biclusters have been used so far (Orzechowski, 2013; Pontes *et al.*, 2015b). Some algorithms yield only a single bicluster at a time, rendering their application cumbersome (Pontes *et al.*, 2015a). Other methods output a high number of biclusters (e.g., BiMax (Prelić *et al.*, 2006) and PBBA (Orzechowski and Boryczko, 2016b)). This usually produces many overlaps and degrades the overall performance of the algorithm (Eren *et al.*, 2013).

Providing the proper balance between local and global context within the data is also difficult. The methods that model global relations are typically able to deliver only a limited number of results (e.g., Plaid (Lazzeroni and Owen, 2002), FABIA (Hochreiter *et al.*, 2010), and ISA (Bergmann *et al.*, 2003)), or tend to exhibit decreased accuracy with each result (e.g., CC (Cheng and Church, 2000)). On the other hand, algorithms that focus on local similarities are susceptible to losing global reference (e.g., Bimax, PBBA, or UniBic (Wang *et al.*, 2016)). For example, UniBic, which sorts pairs of values and column indices of each row in order to identify the longest common subsequences, is able to detect the longest order-preserving pattern between each pair of rows, irrespective of the order of columns, but it fails to capture narrow biclusters containing only a few rows and multiple columns.

As the biclustering problem is NP-hard, designing an efficient and accurate parallel biclustering algorithm remains a challenge. Most of the reference biclustering algorithms are purely sequential. The reason for this is that the methods either require intensive computations, which limit their application to datasets of smaller size, or are fast but at the cost of lower accuracy.

2 Methods

In this paper a novel biclustering algorithm called EBIC is introduced, which overcomes the above shortcomings. The algorithm is based on evolutionary computation, a subfield of Artificial Intelligence (AI). It is likely the first biclustering algorithm capable of detecting all aforementioned types of meaningful patterns with very high accuracy. EBIC is also one of very few parallel biclustering methods. We show that the proposed algorithm outperforms the most established methods in the field with respect to accuracy and relevance on both synthetic and real genomic datasets. An open-source, multi-GPU, parallel implementation of the algorithm is also provided.

The algorithm is designed for environments with at least a single GPU and requires the installation of CUDA. The algorithm was developed in C++11 with OpenMP, with CUDA used for parallelization.

2.1 Motivation.

The design of the algorithm is motivated by the following observation. Given the input matrix $A = \{a_{ij}\}$, where i stands for rows and j for columns, consider counting the number of rows with the property that the value in column p is smaller than the value in column q, i.e. $\#\{k : a_{kp} < a_{kq}\}$. If the values in the dataset are generated randomly with univariate distribution, half of the rows on average are expected to have this property, and half are not. Addition of another column r to the series, such that values in this columns are larger than the values in column q, i.e. $\#\{k : a_{kp} < a_{kq} < a_{kr}\}$, should result in another reduction of the number of rows by half. Thus, for data without any signal, each addition of a column to the series reduces the number of concordant rows by half. On the other hand, if the distribution of the data is not uniform and there exists a monotonic relationships between rows in some subset of conditions, any addition of the pattern-specific column won't eliminate the rows belonging to this pattern. Thus, the algorithm attempts to intelligently manipulate multiple series of columns and assigns higher scores to those series in which column additions do not result in total reduction of the rows.

The quality of each bicluster is determined by a function (called fitness), which takes into consideration the number of columns and, exponentially, the number of rows that follow the monotonically increasing trend represented by each series of columns. The design of the fitness function promotes incorporation of new columns to biclusters, provided there is a sufficient number of rows matching the trend (1).

$$f(B) = \begin{cases} 2^{\min(|I|-\sigma,0)} \cdot |J| \cdot \log(\max(|J|-1,0)) & if\,|J| > 1 \\ 0 & if\,|J| \leq 1\,, \end{cases} \tag{1}$$

where σ is the expected minimal number of rows that should be included within a bicluster $B = (I, J)$, with its rows and columns denoted as I and J, respectively.

EBIC uses a different representation compared with other evolutionary-based biclustering methods (Divina and Aguilar-Ruiz, 2006; Mitra and Banka, 2006; Ayadi *et al.*, 2012). Instead of modeling a bicluster as a tuple with a set of rows and a set of columns, biclusters in EBIC are represented by a series of column indices. The quality of a given series is calculated based on the number of rows that match the monotonous rules present within the series of columns. The modification of column series is performed using an AI-based technique known as genetic programming (GP) (Koza, 1992; Poli *et al.*, 2008b). Series of columns are expanded only when the rule they impose is matched by sufficient number of rows.

EBIC belongs to the family of hybrid biclustering approaches (Orzechowski and Boryczko, 2016a) and features several techniques commonly used in evolutionary algorithms. The development of biclusters is driven by simple genetic operations: 1) four different types of mutations – insertion of a new column to the series (Fig. 1a), deletion of one of the columns from the series (Fig. 1b), swap of two columns within the series (Fig. 1c), and substitution of a column within the series (Fig. 1d); and 2) crossover (Fig. 1e). The individuals that are set to undergo genetic operations are determined using tournament selection. To obtain a diverse set of solutions, a variant of a technique called *crowding* is used, which limits the probability of selecting those individuals that share columns with those already added to the new generation (Sareni and Krahenbuhl, 1998). More specifically, the fitness of individuals that take part in a tournament is decreased by the homogeneous penalty of 1.2^{ϑ}, where ϑ corresponds to the average penalty of using each of the columns separately. The explanation for this value of the parameter is provided in Supplementary Data. The described penalty enhances additions to the population individuals with underrepresented columns, what highly increases the diversity in population.

Individuals whose overall fitness is the highest are stored in the top-rank list, which is updated only if a newly found individual does not substantially overlap with an individual in the list. During the construction of a new population a variant of a *tabu list* is used, which forbids calculation of the previous biclusters (Glover, 1989, 1990). Elitism is used to clone a group of the best individuals found so far, so that the population is still able to

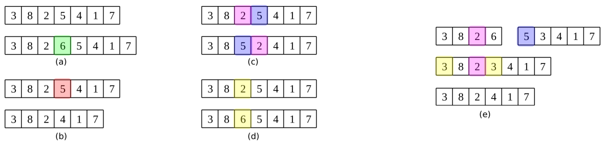

Fig. 1: Genetic operations in EBIC: (a) insertion mutation, (b) deletion mutation, (c) swap mutation, (d) substitution mutation, and (e) crossover.

search around local minima (Poli *et al.*, 2008a). To limit the communication overhead, a Compressed Biclusters Format (CBF) is proposed for storing biclusters (see Fig. 2). The format was motivated by Compressed Row Storage (CRS), a popular representation of sparse matrices.

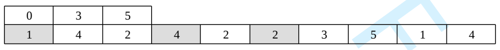

Fig. 2: Compressed Bicluster Format (CBF) uses two arrays. The first array determines the starting positions of each of the biclusters, while the second one holds indexes of columns of biclusters. In this example the population consists of three biclusters (individuals): (1,4,2), (4,2), and (2,3,5,1,4), which start at indices 0, 3, and 5, respectively.

2.2 EBIC Algorithm.

The basic concept of EBIC – a parallel biclustering algorithm based on Artificial Intelligence (AI) – is presented in Figure 3. The dataset is split into equal chunks of data and distributed across multiple GPUs. A population of different series of columns is generated on the CPU, stored in CBF format, and broadcast to multiple GPUs. Each GPU counts the number of rows which match the given series. The results are summarized on each GPU and sent back to the CPU in order to calculate fitness, which is used later to assess bicluster quality.

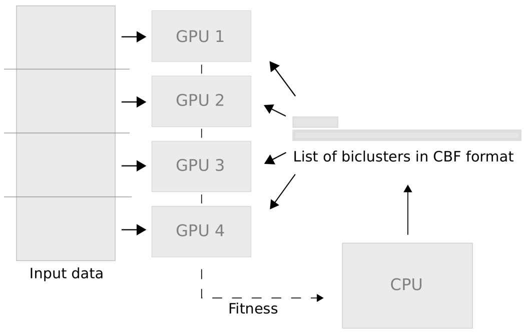

Fig. 3: Overview of EBIC. After dispatching chunks of the input data to multiple GPUs, biclusters – represented by multiple series of columns and stored in CBF format – are broadcast to GPUs. Each GPU calculates the number of how many rows of the chunk match the series imposed by the columns. This is used to determine fitness of each bicluster and generate a new set of biclusters.

Step 1: Initialization. Set up GPUs, divide the dataset proportionally by rows depending on the number of GPUs, and distribute the data across multiple GPUs. Generate initial population, calculate fitness on GPUs. Initialize top-rank list by sequentially adding unique (non-overlapping) series of columns with the highest fitness according to (1).

Step 2: Elitism. Reproduce 1/4 of the best biclusters from the top-rank list, add them to the new population. Update penalties for using each column (each column addition to the population increases the penalty for using this column).

Step 3: Prepare population of biclusters. Until the population reaches its required size, try to generate unique solutions (i.e., that haven't been previously analyzed). Select each new individual using tournament selection. Thus, select a solution randomly from the previous population and adjust its quality by applying the penalty for similarity with the previously accepted solutions. The penalty $vartheta$ is calculated by averaging penalties incurred by selecting each column separately over the number of columns within the series. The final penalty is calculated using the value of 1.2^{ϑ}. After selecting individuals, perform genetic operations (crossover and mutation). If the solution is novel (i.e., does not belong to the tabu list) add it to the population and the tabu list, and update penalties for using the solution's columns. Store the population in Compressed Biclusters Format (CBF). If the solution was previously analyzed, increase the number of tabu-list hits. If this number is greater than the size of population, finish calculations and go to step 6 in order to report the previously found best patterns.

Step 4: Calculate quality of biclusters in parallel. Dispatch the new population (i.e., sets of column series) to each of the GPUs. Determine how many rows match each of the series of columns. Collect the results from multiple GPUs and determine the fitness of each bicluster according to (1).

Step 5: Update top-rank list. Sort the population according to fitness. Try to add new individuals to the top-rank list by checking if they do not substantially overlap with records with higher fitness. If a bicluster is added, remove from the top-rank list all records that have lower fitness and substantially overlap with the bicluster. After all individuals in the population are checked, remove from the top-rank list the records that have the lowest fitness, until the required size of the top-rank list is reached. If the maximal number of iterations is not accomplished, go back to Step 2.

Step 6: Prepare biclusters. Determine in parallel on each GPU the indices of rows that match each of the series of columns in the top-rank list.

Step 7: Expansion of biclusters. Expand the biclusters that have approximate and negative trends. Output the required number of biclusters (or all biclusters from the top-rank list).

2.3 Pattern discovery on synthetic datasets.

The performance of EBIC was evaluated on the benchmark of synthetic datasets from (Wang *et al.*, 2016) and compared to top biclustering methods: UniBic (Wang *et al.*, 2016), OPSM (Ben-Dor *et al.*, 2003), QUBIC (Li *et al.*, 2009), ISA (Bergmann *et al.*, 2003), FABIA (Hochreiter *et al.*, 2010), CPB (Bozdağ *et al.*, 2009), and BicSPAM (Henriques and Madeira, 2014), as well as a newly published GPU-accelerated biclustering algorithm called Condition-dependent Correlation Subgroup (CCS) (Bhattacharya and Cui, 2017). The latter hasn't been benchmarked yet on the established collection of datasets, neither synthetic nor genomic.

The test suite that was used to benchmark the algorithms contains three very popular biclustering problems: pattern discovery, biclusters overlap, and narrow biclusters detection. Recovery and relevance scores were determined using the Jaccard index (Jaccard, 1901) from the BiBench

Table 1. Description of GDS datasets.

Dataset	Genes	Samples	Description
GDS181	12626	84	Large-scale analysis of the human Transcriptome
GDS589	8799	122	Multiple normal tissue gene expression across strains
GDS1406	12488	87	Brain regions of various inbred strains
GDS1451	8799	94	Toxicants effect on liver: pooled and individual sample comparison
GDS1490	12488	150	Neural tissue profiling
GDS2520	12625	44	Head and neck squamous cell carcinoma
GDS3715	12626	110	Insulin effect on skeletal muscle
GDS3716	22283	42	Breast cancer: histologically normal breast epithelium

package (Eren *et al.*, 2013), specifically (2) and (3) :

$$Recovery = \sum_{e \in expected} max_{f \in found} \frac{|e \cap f|}{|e \cup f|} \qquad (2)$$

$$Relevance = \sum_{f \in found} max_{e \in expected} \frac{|e \cap f|}{|e \cup f|} \qquad (3)$$

The first set of problems verifies the ability of the algorithm to identify six different data patterns, including trend-preserving, column-constant, row-constant, shift, scale, and shift-scale. The tests assess how accurately a biclustering algorithm detects three biclusters of size 15x15 implanted within a matrix of size 150x100, four biclusters of size 20x20 implanted within a matrix of size 200x150, and five biclusters of size 25x25 implanted within a matrix of size 300x200. Each problem consists of 5 different datasets for each of 6 patterns – which constitute 90 unit tests in total. The tests on overlapping patterns measure the ability of the algorithms to detect 5 biclusters of size 20x20 implanted within the matrix of size 200x150 that overlapped with each other by 0x0, 3x3, 6x6, and 9x9 elements – 20 tests in total (Wang *et al.*, 2016). Narrow biclusters are biclusters with 100 rows and 10–30 columns implanted within a large matrix of size 1000x100 – 9 tests in total. The tests determine whether biclustering methods are capable of discovering patterns that feature multiple rows but only a small number of columns (Wang *et al.*, 2016). To show independence of the results, our method was run 10 times on all problems. Each time, a different seed served to initialize a pseudo-random number generator, which was used to initialize the population in the first iteration.

2.4 Enrichment analysis on genomic datasets.

The effectiveness of pattern discovery with EBIC was further evaluated on real-world gene expression datasets. For this purpose, BiBench software and a benchmark of genetic datasets from Eren et al. (Eren *et al.*, 2013) were used. Details of the gene datasets used for the study are presented in Table 1. The same procedures of data acquisition, preprocessing, and analysis were followed. Thus, datasets were downloaded using GEOquery (Davis and Meltzer, 2007) and preprocessed using PCA imputation (Stacklies *et al.*, 2007). After completing biclustering, a gene enrichment analysis of each bicluster was performed using the R package GOstats (Falcon and Gentleman, 2007). Biclusters were considered significantly enriched if any of the p-values associated with a given GO term were lower than 0.05 after Benjamini-Hochberg correction (Benjamini and Hochberg, 1995). Assessment of the results was based on the proportion of enriched biclusters to all biclusters reported. Each algorithm was allowed to return no more than 100 biclusters per dataset. The number of biclusters found and the proportion of significantly enriched results were compared to the study by (Wang *et al.*, 2016) and are presented in Table 2. EBIC was tested with two overlap ratios, 0.5 and 0.75.

3 Results

The performance of EBIC was tested on both synthetic as well as real gene expression datasets. Synthetic benchmark from Wang et al. (Wang *et al.*, 2016) is available at `https://sourceforge.net/projects/unibic/files/data_result.zip`. For biological validation a well-established benchmark from Eren et al. was used (Eren *et al.*, 2013) with eight genetic datasets. The collection of datasets and the results of EBIC on both synthetic and genetic datasets could be found in Supplementary Data.

3.1 Pattern discovery on synthetic datasets.

For synthetic datasets EBIC was set to stop either after 20,000 iterations or when the number of tabu-list hits exceeded the size of the population. All parameters were set to their defaults. Columns of biclusters were allowed to overlap no more than 0.5, and the block-size for the CUDA kernel was set to 64. This took a reasonable amount of computation time (1-25 minutes on Intel Core i7-6950X CPU with GeForce GTX 1070 GPU). Comparison of the accuracy of EBIC with selected biclustering methods in terms of recovery and relevance is presented in Fig. 4. CCS did not manage to return any result for trend-preserving, and row- and column-constant patterns, thus the method was excluded from the comparison. The CCS algorithm managed to present partial solutions for shift-, scale-, and shift-scale patterns only.

The average recovery and relevance scores of EBIC are better than those reported by any of the previous methods. This difference is especially visible in order-preserving and shift-scale problems, which are considered to be the most biologically meaningful (Wang *et al.*, 2016). EBIC managed to detect all patterns perfectly for trend-preserving patterns, while other methods reached 70% on average. The average relevance and recovery rate for shift-scale patterns were also much higher. As for scale and shift-scale patterns, EBIC attained high recovery/relevance scores across all tests (95.2%/85.5% for scale- and 94.2%/84.5% for shift-scale patterns), although scores for the worst-case scenarios were comparable to other methods (75.1%/37.8% and 72.0%/46.7%, respectively). EBIC may be the first biclustering algorithm capable of detecting all aforementioned patterns with over 90% average recovery and relevance (Pontes *et al.*, 2015a). The recovery/relevance scores from multiple runs of the algorithm initialized with different random numbers did not differ statistically.

EBIC was also tested on the datasets provided by Bhattacharya et al. (Bhattacharya and Cui, 2017) and detected biclusters with recovery and relevance scores over 95%, whereas CCS reported those scores to vary from approximately 20% to nearly 90%.

Overlapping biclusters. The second set of tests compares the deterioration of the accuracy of biclustering algorithms in detecting trend-preserving biclusters that overlap with each other. This set of problems contains tests of 3 biclusters of size 20x20 that overlap with each other by 0x0, 3x3, 6x6, and 9x9 within a matrix of size 200x150. Each problem is represented by 5

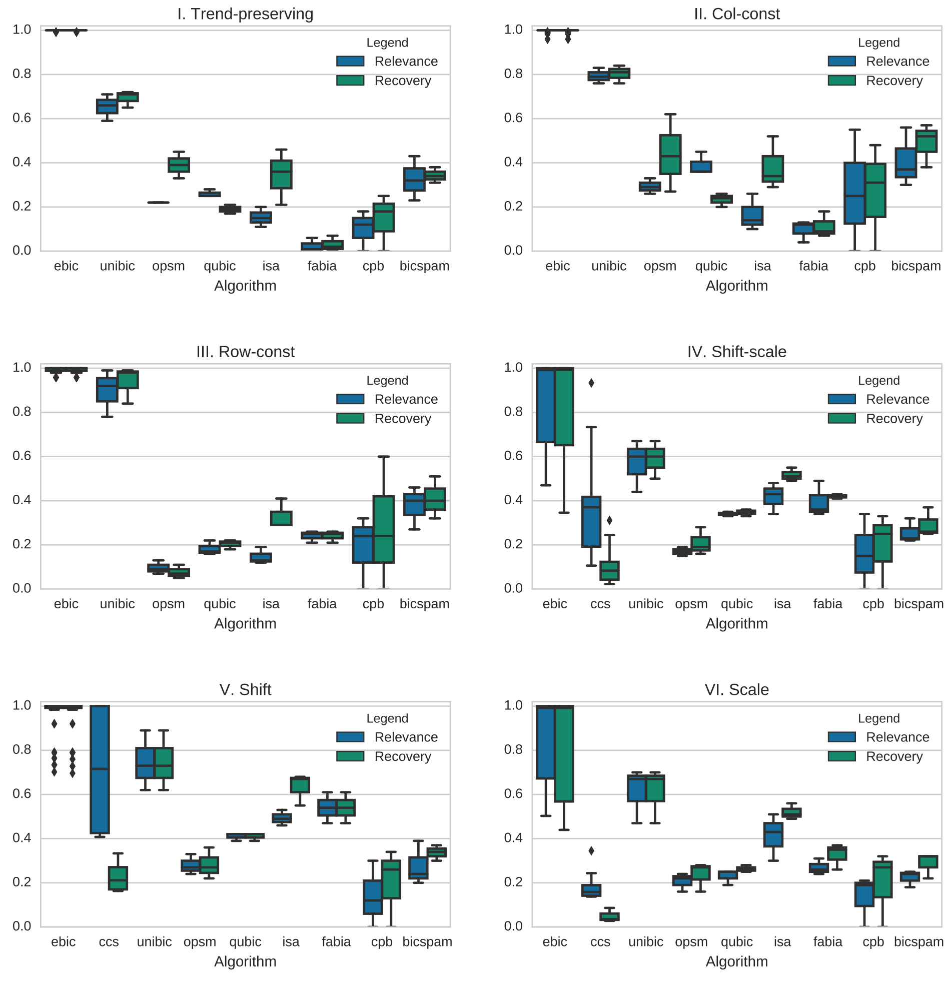

Fig. 4: Comparison of the performance of biclustering algorithms on different types of patterns. Scores of the algorithms other than EBIC and CCS are quoted from (Wang *et al.*, 2016).

dataset variants, resulting in up to 20 tests in total (Wang *et al.*, 2016). The effect of the overlap on the recovery and relevance of different algorithms is presented in Fig. 5.

All biclustering methods tend to deteriorate if implanted biclusters start to significantly overlap with each other (Wang *et al.*, 2016). The performance of EBIC also decreased when the higher level of overlap was considered, but the decrease was small. The algorithm was still able to maintain recovery and relevance scores close to 90% on the average. The second-best method was UniBic, which deteriorated from around 90% for non-overlapping biclusters down to 60% recovery and 85% relevance for the most overlapping structures.

Narrow biclusters. The last phase of our benchmark considers the detection of narrow biclusters comprising 100 rows and 10/20/30 columns, which were implanted within the matrix of size 1000x100. Each scenario contains 3 variants, resulting in up to 9 tests in total. The results are presented in Fig. 6.

In contrast to all other algorithms, EBIC managed much better in this task and discovered almost perfectly all implanted structures. For the narrowest biclusters, our algorithm was approximately twice as good as the second method dedicated to finding narrow biclusters (BicSPAM). UniBic was reported to have low accuracy in detecting narrow biclusters within the dataset. CCS did not manage to return any bicluster for every dataset in this test.

Noise sensitivity. Noise sensitivity analysis of EBIC may be found in Supplementary Data. Tuning of EBIC parameters allows the method to be reasonably resistant to up to $N(0, 0.25)$ of normally distributed noise.

Summary. Our general conclusion is that EBIC is not only capable of detecting different types of patterns, but also different sizes of patterns (i.e., wide or narrow patterns) with very high accuracy.

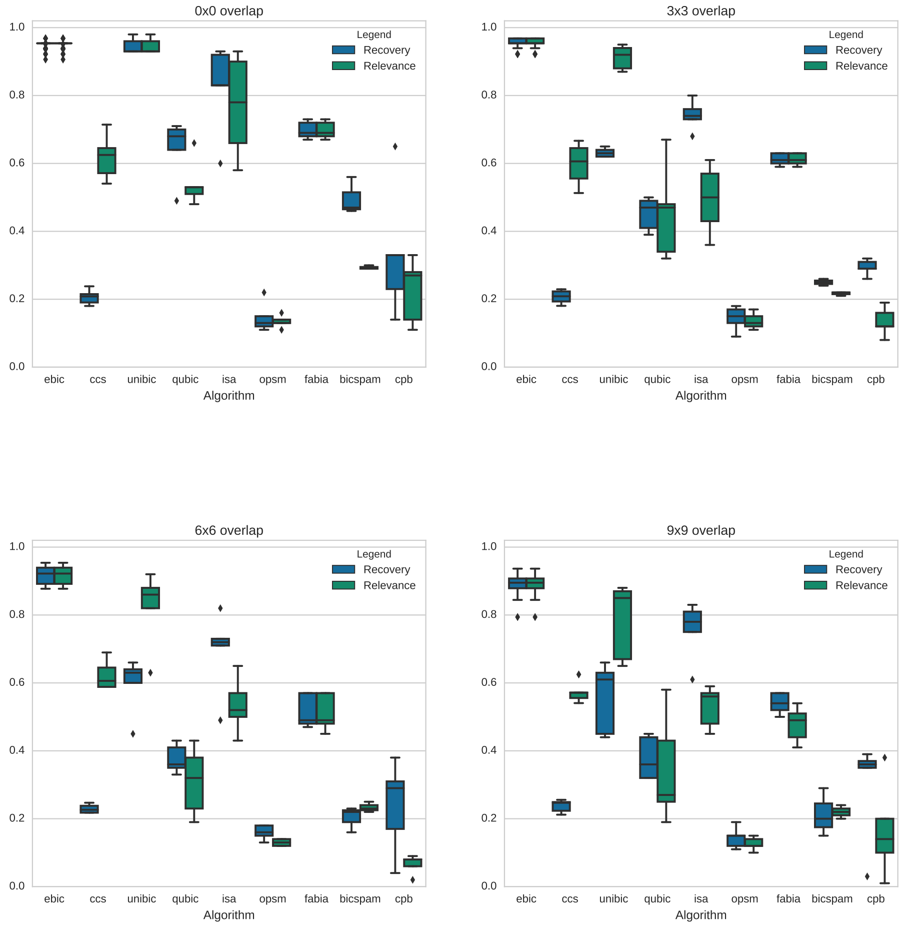

Fig. 5: Comparison of the performance of biclustering algorithms in scenarios with different levels of biclusters' overlap. Scores of the algorithms other than EBIC and CCS are quoted from (Wang *et al.*, 2016).

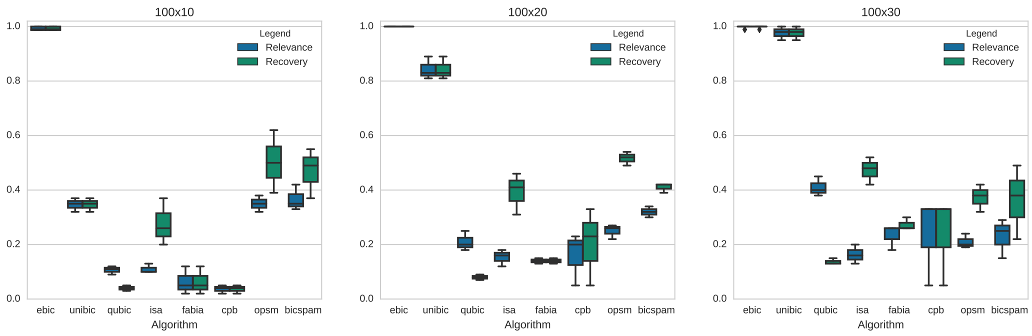

Fig. 6: Comparison of biclustering-algorithm performance in scenarios with narrow biclusters. The reference results are quoted from Wang et al.(Wang *et al.*, 2016).

3.2 Enrichment analysis on genomic datasets.

For the genetic datasets, it was observed that the proportion of enriched biclusters obtained after approximately 5000 iterations highly depended on the dataset (see Supplementary Data). Further iterations either improved or worsened the proportion. EBIC was run for 5000 iterations, columns were allowed to overlap by 50% or 75%. The results of enrichment analyses are presented in Table 2.

Some memory management issues were encountered with CCS (both the sequential and parallel versions). The algorithm was unable to detect biclusters in some of the genomic datasets and terminated prematurely with an error. After fixing a bug, the algorithm, even in parallel mode, proved to be extremely slow. Although the dataset was of reasonable size it took over 8 days of computation (on CPU+GPUs) to yield results for the most challenging genomic datasets (GDS 1451). In contrast EBIC needed less than 3 minutes to yield higher number of significantly enriched biclusters for this dataset.

EBIC generated the highest percentage of enriched biclusters. EBIC with a more restrictive overlap ratio (0.5) generated a higher percentage of significantly enriched biclusters (52.4%) in comparison to any other method. The second best was CCS (43.5%), which on the other hand generated much more significantly enriched biclusters. EBIC with less restrictive overlap (0.75) outperformed all the methods included in our study, both in terms of the number and percentage of significantly enriched biclusters. EBIC generated 20 significantly enriched biclusters more than the second-best method (323 vs 303 by CCS). More importantly, EBIC managed to find nearly 11% more significantly enriched biclusters. This result is noticeable, considering that the difference between the second- and third-best methods was only 2.7%. In addition, the biclusters returned did not overlap substantially, from less than 4% up to 31%, depending on the dataset. The datasets as well as the results of biological validation of EBIC and CCS are available in Supplementary Data.

Table 2. Significantly enriched biclusters found across all GDS datasets. Two overlap thresholds of EBIC are considered: 0.5 and 0.75. The scores of the algorithms other than EBIC and CCS are quoted from (Wang et al., 2016).

Algorithm	Found	Enriched
EBIC, 0.75	**589**	**323 (54.8%)**
EBIC, 0.5	145	76 (52.4%)
CCS	691	303 (43.8%)
UniBic	151	62 (41.1%)
OPSM	163	48 (29.5%)
QUBIC	91	34 (37.4%)
ISA	217	71 (32.7%)
FABIA	80	22 (27.5%)
CPB	96	34 (35.4%)

We reinspected the results of the two best methods (EBIC-0.75 and CCS) after applying a filtering proposed by Prelic et al. (Prelić *et al.*, 2006) and implemented in Eren et al. (Eren *et al.*, 2013). The procedure removed biclusters that overlap with the others by over 25%. After filtering, 296 out of 589 biclusters for EBIC remained, out of which 122 were found to be significantly enriched (41.2%). For CCS, 332 out of 619 remained and only 113 were marked as significantly enriched (34.0%). We eschewed testing the other methods, as their number of significantly enriched biclusters before the filtering procedure was even applied was lower than the one for EBIC or CCS after applying the procedure.

3.3 Scalability of the algorithm.

In order to assess the scalability of the methods, five datasets with 100 columns and different numbers of rows ranging between 5000 and 25000 were generated. Times were averaged based on five runs of the methods on each of the datasets with their default parameters. The algorithms were allowed to yield up to 100 biclusters. All tests were performed on a machine with an Intel Core^{TM} i7-6950X CPU and 64GB of RAM. Comparison of run times in logarithmic scale is presented in Fig. 7. Starting with 10,000 rows, EBIC began to run faster than both CCS and UniBic, the most precise methods so far. For problems with 25,000 rows, EBIC was over 12 times faster than UniBic and over 20 times faster than CCS. With increasing data size, the running times of EBIC have started to be comparable with ones from OPSM and ISA. The actual performance of EBIC for larger datasets on multiple GPUs requires further investigation.

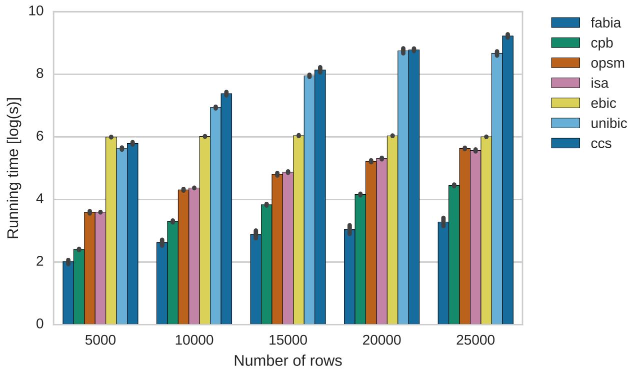

Fig. 7: Comparison of running time of the algorithms on datasets with 100 columns and varying numbers of rows.

A complexity analysis of EBIC can be found in Supplementary Data.

4 Discussion

EBIC is one of the very few parallel biclustering methods dedicated for multi-GPU environments. In comparison with state-of-the-art algorithms EBIC exhibited a number of advantages: (1) EBIC outperformed the state-of-the-art biclustering algorithms on established synthetic datasets. EBIC was the only algorithm to discover each of six types of major genetic patterns in synthetic datasets with over 95% average accuracy and the only one to maintain over 90% accuracy on narrow and overlapping biclusters. (2) EBIC found over 11% more significantly enriched biclusters than the second-best method (CCS) on a benchmark of 8 genomic datasets (over 7% more after removing overlapping biclusters). (3) EBIC yielded far more significantly enriched biclusters than any of the methods (even after removing overlapping biclusters). (4) EBIC proved to be over 12 times faster than any of the most accurate methods (CCS or UniBic) on the largest datasets.

We would like to formulate the requirements for the next-generation of biclustering methods. Such algorithms are expected to meet the following criteria: (1) be capable of discovering the six major types of biclusters discussed above with high accuracy (over 75% on average); (2) be capable of handling overlapping, narrow, and approximate patterns with similar accuracy; (3) provide meaningful solutions for both synthetic and real datasets; (4) be scalable. In contrast to other methods described in this paper, EBIC with its average accuracy exceeding 90% certainly meets these requirements and could be called a next-generation biclustering method.

EBIC has certain limitations. First, the closer the overlap threshold to 0, EBIC may no longer be able to capture different series that are present within the same columns. Instead, this series of columns which is represented by the largest number of rows will incorporate all other permutations. The reason for this is construction of top-rank list. For performance purposes, the list uses intersection of columns as the merging criterion, what makes the actual order of columns within the series irrelevant. A full overlap of biclusters within the top-rank list is possible, but discouraged. Secondly, application of EBIC to datasets that have fewer than 20 columns is discouraged. In this case an exhaustive search guarantees discovery of all meaningful patterns in a much shorter time. Thirdly, the overlap degree of biclusters for a dataset requires verification. Tuning the parameters of the method may decrease the level of overlaps. A more restrictive overlap threshold (0.5) allows the algorithm to detect fewer biclusters with less overlapping columns, while a less restrictive overlap threshold (0.75) returns more biclusters at the cost of their overlap. The degree of biclusters' overlap cannot be directly controlled in EBIC.

The guidelines for the exact number of iterations to run EBIC, as well as the optimal level of overlap on biclusters in the top-list, need to be empirically defined. EBIC scores do not seem to improve with every iteration. The accuracy of pattern detection generally improves over time for synthetic datasets, but this did not hold for real genomic datasets. The highest proportion of significantly enriched biclusters oscillated or even slightly deteriorated for real-world genetic datasets after 100 iterations. For all genomic datasets, EBIC was stopped after 5,000 iterations, as it seemed to be a reasonable compromise between the percentage of enriched results and run time. Additional study on the influence of the size of the input matrix on the number of required iterations is needed.

Our initial tests using larger volumes of data indicate that the algorithm supports datasets of up to 60k rows per GPU. Full scalability of EBIC and preparing the algorithm for big data challenges requires more work.

5 Conclusions

EBIC is anticipated to become a reference method for future studies in biclustering. EBIC may also prove beneficial in other domains beyond genomics. The method may improve pattern detection in multiple other fields (e.g. medicine, applied informatics, economics, biology, or chemistry) in which biclustering has been previously successfully applied. Extensive AI method development is necessary to fully realize the potential of AI for solving the most challenging big data problems.

Authors' contribution

P.O. conceived the study, designed and implemented the algorithm. P.O. and J.H.M. performed the analysis. M.S. and X.H. consulted the project, analyzed the results and participated in writing the manuscript. J.H.M. oversaw the project.

Acknowledgements

This research was supported in part by PL-Grid Infrastructure and by National Institutes of Health grants LM012601, TR001263, and ES013508.

References

Ayadi, W., Maâtouk, O., and Bouziri, H. (2012). Evolutionary biclustering algorithm of gene expression data. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*, pages 206–210. IEEE.

Ben-Dor, A., Chor, B., Karp, R., and Yakhini, Z. (2003). Discovering local structure in gene expression data: the order-preserving submatrix problem. *J. Comput. Biol.*, **10**(3-4), 373–384.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300.

Bergmann, S., Ihmels, J., and Barkai, N. (2003). Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical review E*, **67**(3), 031902.

Bhattacharya, A. and Cui, Y. (2017). A gpu-accelerated algorithm for biclustering analysis and detection of condition-dependent coexpression network modules. *Scientific Reports*, **7**(1), 4162.

Bozdağ, D., Parvin, J. D., and Catalyurek, U. V. (2009). A biclustering method to discover co-regulated genes using diverse gene expression datasets. In *Bioinformatics and Computational Biology*, pages 151–163. Springer.

Busygin, S., Prokopyev, O., and Pardalos, P. M. (2008). Biclustering in data mining. *Computers & Operations Research*, **35**(9), 2964–2987.

Cheng, Y. and Church, G. M. (2000). Biclustering of expression data. In *Proceedings of the eighth international conference on intelligent systems for molecular biology*, volume 8, pages 93–103.

Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., Ferrero, E., Agapow, P.-M., Xie, W., Rosen, G. L., *et al.* (2017). Opportunities and obstacles for deep learning in biology and medicine. *bioRxiv*, page 142760.

Davis, S. and Meltzer, P. S. (2007). Geoquery: a bridge between the gene expression omnibus (geo) and bioconductor. *Bioinformatics*, **23**(14), 1846–1847.

Divina, F. and Aguilar-Ruiz, J. S. (2006). Biclustering of expression data with evolutionary computation. *IEEE transactions on knowledge and data engineering*, **18**(5), 590–602.

Dolnicar, S., Kaiser, S., Lazarevski, K., and Leisch, F. (2012). Biclustering: Overcoming data dimensionality problems in market segmentation. *Journal of Travel Research*, **51**(1), 41–49.

Eren, K., Deveci, M., Küçüktunç, O., and Çatalyürek, Ü. V. (2013). A comparative analysis of biclustering algorithms for gene expression data. *Briefings in bioinformatics*, **14**(3), 279–292.

Falcon, S. and Gentleman, R. (2007). Using gostats to test gene lists for go term association. *Bioinformatics*, **23**(2), 257–258.

Glover, F. (1989). Tabu search - part i. *ORSA Journal on computing*, **1**(3), 190–206.

Glover, F. (1990). Tabu search - part ii. *ORSA Journal on computing*, **2**(1), 4–32.

Hartigan, J. A. (1972). Direct clustering of a data matrix. *Journal of the american statistical association*, **67**(337), 123–129.

Henriques, R. and Madeira, S. C. (2014). Bicspam: flexible biclustering using sequential patterns. *BMC bioinformatics*, **15**(1), 130.

Hochreiter, S., Bodenhofer, U., Heusel, M., Mayr, A., Mitterecker, A., Kasim, A., Khamiakova, T., Sanden, S. V., Lin, D., Talloen, W., *et al.* (2010). FABIA: factor analysis for bicluster acquisition. *Bioinformatics*, **26**(12), 1520–1527.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, **37**, 547–579.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Lazzeroni, L. and Owen, A. (2002). Plaid models for gene expression data. *Statistica sinica*, pages 61–86.

Li, G., Ma, Q., Tang, H., Paterson, A. H., and Xu, Y. (2009). QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research*, **37**(15), e101–e101.

Madeira, S. C. and Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, **1**(1), 24–45.

Mirkin, B. (1996). Mathematical classification and clustering.

Mitra, S. and Banka, H. (2006). Multi-objective evolutionary biclustering of gene expression data. *Pattern Recognition*, **39**(12), 2464–2477.

Morgan, J. N. and Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, **58**(302), 415–434.

Orzechowski, P. (2013). Proximity measures and results validation in biclustering–a survey. In *International Conference on Artificial Intelligence and Soft Computing*, pages 206–217. Springer.

Orzechowski, P. and Boryczko, K. (2016a). Hybrid biclustering algorithms for data mining. In *European Conference on the Applications of Evolutionary Computation*, pages 156–168. Springer.

Orzechowski, P. and Boryczko, K. (2016b). Propagation-based biclustering algorithm for extracting inclusion-maximal motifs. *Computing & Informatics*, **35**(2).

Padilha, V. A. and Campello, R. J. (2017). A systematic comparative evaluation of biclustering techniques. *BMC bioinformatics*, **18**(1), 55.

Poli, R., McPhee, N. F., and Vanneschi, L. (2008a). Elitism reduces bloat in genetic programming. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1343–1344. ACM.

Poli, R., Langdon, W. B., McPhee, N. F., and Koza, J. R. (2008b). *A field guide to genetic programming*. Lulu.com.

Pontes, B., Giráldez, R., and Aguilar-Ruiz, J. S. (2015a). Biclustering on expression data: A review. *Journal of biomedical informatics*, **57**, 163–180.

Pontes, B., Girldez, R., and Aguilar-Ruiz, J. S. (2015b). Quality measures for gene expression biclusters. *PloS one*, **10**(3), e0115497.

Prelić, A., Bleuler, S., Zimmermann, P., Wille, A., Bühlmann, P., Gruissem, W., Hennig, L., Thiele, L., and Zitzler, E. (2006). A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, **22**(9), 1122–1129.

Sareni, B. and Krahenbuhl, L. (1998). Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation*, **2**(3), 97–106.

Stacklies, W., Redestig, H., Scholz, M., Walther, D., and Selbig, J. (2007). pcamethods - a bioconductor package providing pca methods for incomplete data. *Bioinformatics*, **23**(9), 1164–1167.

Wang, Z., Li, G., Robinson, R. W., and Huang, X. (2016). Unibic: Sequential row-based biclustering algorithm for analysis of gene expression data. *Scientific reports*, **6**.