Computing and Informatics, Vol. 35, 2016, 391–410

PROPAGATION-BASED BICLUSTERING ALGORITHM FOR EXTRACTING INCLUSION-MAXIMAL MOTIFS

Patryk Orzechowski

AGH University of Science and Technology, Kraków
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Automatics and Biomedical Engineering
e-mail: patrick@agh.edu.pl

Krzysztof Boryczko

AGH University of Science and Technology, Kraków
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
e-mail: boryczko@agh.edu.pl

Abstract. Biclustering, which is simultaneous clustering of columns and rows in data matrix, became an issue when classical clustering algorithms proved not to be good enough to detect similar expressions of genes under subset of conditions. Biclustering algorithms may be also applied to different datasets, such as medical, economical, social networks etc. In this article we explain the concept beneath hybrid biclustering algorithms and present details of propagation-based biclustering, a novel approach for extracting inclusion-maximal gene expression motifs conserved in gene microarray data. We prove that this approach may successfully compete with other well-recognized biclustering algorithms.

Keywords: Biclustering, bioinformatics, pattern matching, data mining, microarray gene expression data, conserved gene expression motifs

1 INTRODUCTION

Discovering group of genes responsible for different processes remains one of the challenges in genetics. DNA microarrays usually comprise expression levels of thousands of genes under hundreds of conditions. The values of gene expression data are real numbers, usually the logarithms of the relative abundance of gene's mRNA under the specified condition. Similar expression patterns of genes are evidence of their similar function. Isolated gene does not control any biological function by itself, usually a set of genes is responsible for its management.

Biclustering is one of data mining techniques that proves to be extremely useful in detecting local patterns in gene microarray datasets. In contrast to classical clustering algorithms, biclustering algorithms are designed to take into account rows and columns simultaneously. The object of interest are local patterns, i.e. gene expression level under subset (but not all) conditions [27]. Clustering algorithms may only distinguish genes by their response to all conditions, or differentiate conditions, by analysing whole gene expression profile.

Several studies have already been conducted in the subject of biclustering [23]. Different bicluster types and naming conventions have been proposed in [18]. Detailed comparison of algorithms and their applications to specific problems has been conducted in [5]. Out of multiplicity of biclustering methods, sole basis of biclustering could be narrowed to the following ones: Cheng and Church [4], Plaid [16], SAMBA [31], OPSM [2], xMotif [21], Spectral Biclustering [15], ISA [14], Bimax [27], BBC [9], QUBIC [17] and FABIA [11].

In this article, we present a novel biclustering technique that combines strengths of two algorithms: speed of Bimax and applicability of xMotif. The proposed method, based on propagation of information through data matrix, has been successfully applied to a couple of biological [24] and text datasets [25]. Our method may be considered as a basis for further experiments that will improve biclustering quality by omitting (substantially overlapping) biclusters. The flexibility of the approach allows to construct more advanced methods tailored to specific proposes that could challenge well-esteemed biclustering methods.

2 MOTIVATION

With multiplicity of existing biclustering approaches almost every novel biclustering algorithm to some extent retains similarity to the previously developed methods. This resemblance usually involves reuse of existing mechanisms (i.e. classifiers, metrics etc.) or adaptation of some concepts (such as techniques, motivation) of other algorithms, but in an unusual manner. Thus, many novel solutions are inspired by existing ones and merge underneath some of their strategies.

Realizing this fact lied upon the emergence of hybrid biclustering algorithms. These are the methods which reinvent the development process of new algorithms, as they require understanding the benefits and drawbacks of each of the components

of merged mechanisms beforehand. Combining existing mechanisms requires also attention, as the resulting algorithm doesn't necessarily lead to better solutions.

One of the representatives of hybrid biclustering algorithms is Propagation-Based Biclustering Algorithm (PBBA), which has been proposed herein. The algorithm combines two different approaches: detecting a similar pattern for a set of genes, called xMotif [21], and finding inclusion-maximal biclusters [27]. Our approach may be considered as an aggregated approach, as it uses in a some of features of both algorithms.

The definition and classification of biclusters as well as background for both algorithms (xMotif and Bimax) are presented in this section.

2.1 Biclustering

Biclustering problem may be formulated as finding one (or a couple) of biclusters in data matrix that meet specific homogeneity criterion. (Definition 1).

Definition 1 (Definition of bicluster)**.** For a given dataset $A = \{a_{ij}\}_{m \times n}$ with rows $X = \{x_{1,}, \ldots, x_n\}$ and columns $Y = \{y_{1,}, \ldots, y_m\}$, subset of rows I and subset of columns J is called a bicluster $B = (I, J) = \{a_{ij} \in A : i \in I, j \in J\}$ if $I \subseteq X$ and $J \subseteq Y$ and rows I are similar across columns J according to a homogeneity criterion.

Biclustering algorithms are usually designed to identify one or more of the following major classes of biclusters [18]:

Constant biclusters. All values within the bicluster are exactly the same.

Biclusters with constant rows (columns). In each row (column) of bicluster values are the same. Values in different rows (columns) may differ.

Coherent biclusters (additive model). Values in each row (column) are obtained by adding a constant to some other row (column).

Coherent biclusters (multiplicative model). Values in each row (column) are obtained by multiplying some other row (column) by a constant value.

Biclusters with coherent evolutions. The elements of matrix are considered only as categorical variables, coherent biclusters are identified regardless of the same values in data matrix.

The aim of biclustering is to obtain a single or set of biclusters at a time. Biclusters returned at each run of algorithm (or during a single run) may overlap with each other (on rows, columns or both) or are completely separated [18]. An acceptable compromise improving relevance of the results of Bimax algorithm allows 25 % overlap between biclusters [5].

2.2 Conserved Gene Expression (xMotif) Algorithm

Conserved gene expression motif or *xMotif* proposed by Murali may be considered as a synonym for a bicluster, which is described as subset of genes simultaneously conserved across a subset of samples [21].

Definition 2 (xMotif algorithm)**.** Bicluster $B = (R, C)$ needs to satisfy all the following conditions in order to be called xMotif:

Size. Bicluster B needs to contain at least $\alpha\,\%$ of all columns (i.e. at least αm columns), where $\alpha > 0$.

Conservation. Every row R in bicluster B follows the same motif, i.e. has the same value in each column C in bicluster. Values in different columns may differ.

Maximality. Outside the bicluster B, the motif may be localized in at most β percent of the number of columns in B, i.e. in subset of at most βC size, where $\beta < 1$.

The xMotif algorithm assumes a uniform distribution of data and uses the probabilistic approach to find for each gene statistically significant set of states. After randomly picking *seed* column and *discriminating set* of columns, the algorithm iteratively searches for rows that are in the same state in both *seed* and *discriminating set* [21].

In its original form, xMotif algorithm is designed to identify a coherent evolution of rows [18]. It finds constant values on rows as well [5].

2.3 Bimax Algorithm

Bimax algorithm uses divide-and-conquer strategy. Input matrix is discretized (usually with threshold equal to the mean of data [27]) and then considered as adjacency matrix of bipartite graph consisting of columns and rows. The method finds all inclusion-maximal biclusters, i.e. that cannot be included in any other bicluster (refer to Definition 3 for details).

Definition 3 (Inclusion maximal bicluster)**.** The pair of rows R and columns C forming a bicluster $B = (R, C) \in 2^{1,\ldots,n} \times 2^{1,\ldots,m}$ within binary matrix $A = [a_{ij}]_{m \times n}$ is called *an inclusion-maximal bicluster* if and only if $\forall i \in R, j \in C : a(i, j) = 1$ and $\nexists (R', C') \in 2^{1,\ldots,n} \times 2^{1,\ldots,m} : \forall i' \in R', j' \in C' : a(i', j') = 1 \wedge R' \supseteq R \wedge C' \supseteq C \wedge (R', C') \neq (R, C)$

Bimax algorithm divide set of columns according to first pattern to C_u and C_v. Then rows are resorted to form matrix U of rows responding only to conditions within C_u (and not within C_v), and matrix V with rows responding to conditions within both C_u and C_v plus rows responding to conditions C_v only. After decomposition, the algorithm is executed recursively for matrices U and V (Figure 1).

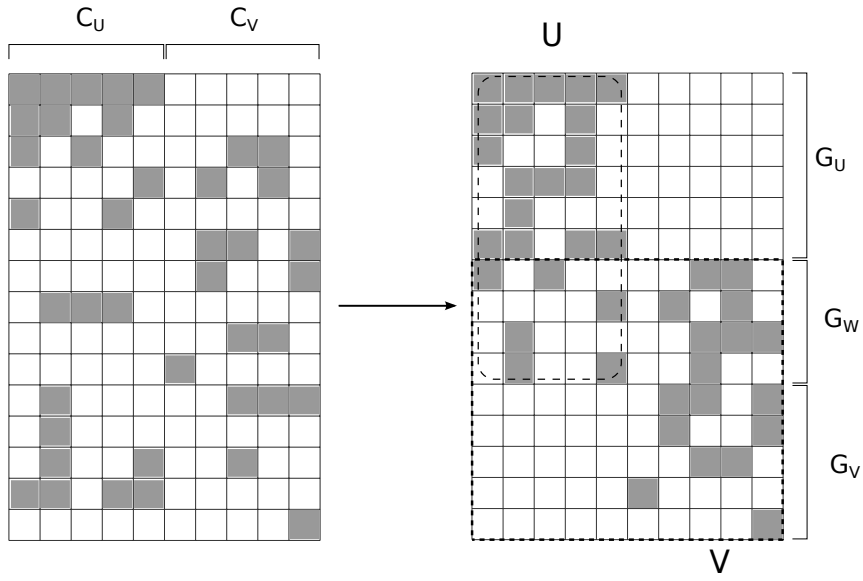

Figure 1. Schema of Bimax algorithm. Discretized matrix A is divided into two matrices U and V on which the algorithm is executed recursively

Bimax is limited to finding only constant upregulated biclusters. It has a reasonable running-time complexity of order $O(nm\beta \min(n, m))$, where β is number of all inclusion-maximal biclusters in A. Bimax is considered as fast and more noise resilient than other biclustering algorithms [5]. Yet, there are some limitations concerning its application.

Low discretization parameter value may result in excessive computation time and producing millions of biclusters [6]. No filtering procedure results in many spurious and overlapping biclusters that would impoverish the relevance score.

3 PROPOSED APPROACH

In this section, we expand Definition 3 of inclusion-maximal bicluster and introduce preprocessing technique called *vicinity transformation*. Properties of *vicinity transformation* are presented as well as definition of propagation-based biclustering is proposed.

3.1 Pattern-Inclusion-Maximal Biclusters

Our model adapts and extends definition of *inclusion-maximal bicluser* proposed by Prelić [27]. Noteworthy, that *inclusion-maximal bicluster* is only a special case of *pattern-inclusion-maximal bicluster*, when all columns of bicluster have equal values to each other.

Definition 4 (Pattern inclusion-maximal bicluster)**.** The pair of rows R and the pair of columns C forming a bicluster $B = (R, C) \in 2^{1,\ldots,n} \times 2^{1,\ldots,m}$ within real matrix $A = [a_{ij}]_{m \times n}$ is called *an pattern inclusion-maximal bicluster* with respect to pattern $v = (v_1, v_2, \ldots, v_m)$ if and only if $\forall i \in R, j \in C : a_{ij} = v_i$ and $\nexists (R', C') \in 2^{1,\ldots,n} \times 2^{1,\ldots,m} : \forall i' \in R', j' \in C' : a_{i'j'} = v_{i'} \wedge R' \supseteq R \wedge C' \supseteq C \wedge (R', C') \neq (R, C)$, where $v_i, v_{i'} \in 2^{1,\ldots,m}$.

In other words, Definition 4 ensures that *pattern inclusion-maximal bicluster* B cannot be further expanded by adding a new column with equal value in every row of the considered bicluster, nor by adding a new row, such that all its values remain equal to the values in the pattern (i.e. values in columns remain equal in each row). Moreover, it guarantees that each *pattern inclusion-maximal bicluster* $B = (R, C)$ is not a subset of any other inclusion maximal bicluster $B' = (R', C')$ (i.e. intersections of rows of biclusters B and B' as well as intersection of columns of biclusters B and B' are not empty).

3.2 Vicinity Transformation

Vicinity transformation is a conversion of input matrix, which locates similar elements across rows (columns). The approach was inspired by neural networks, maximal flow problem [7], water flow simulations [32] and associative artificial intelligence [12, 13].

Definition 5 (Vicinity transformation)**.** *Vicinity transformation* v of a given matrix of integers $A = [a_{ij}]_{m \times n}$ to an integer matrix $B = [b_{ij}]_{m \times n}$ is defined as follows:

$$b_{ij} = v(a_{ij}) = \begin{cases} \max_k \{k : k < i : a_{kj} = a_{ij}\}, & a_{ij} = s_j \wedge \exists k : 0 < k < i : a_{kj} = s_j \\ i, & a_{ij} = s_j \wedge \nexists k : 0 < k < i : a_{kj} = s_j \\ 0, & a_{ij} = 0. \end{cases} \tag{1}$$

where $s = (s_1, s_2, \ldots, s_m)$ – is set of values characteristic for row i motif.

For each non-zero element a_{ij} vicinity transformation of the element $b_{ij} = v(a_{ij})$ returns the index of the nearest row $k : k \leq i$ with non-zero element a_{kj} occurring in the same column j (or its own index i, if such an element does not exist). Schema of vicinity transformation is presented in Figure 2.

Theorem 1 (Properties of vicinity transformation)**.** The list of the following properties results straightforwardly from the definition of vicinity transformation:

Column independence. Vicinity transformation performed on the set of columns simultaneously is equal to performing vicinity transformation on each column separately.

Non-negative. Vicinity transformation of each non-zero element a_{ij} is positive, and for element $v(a_{ij}) = 0$ is equal to zero if and only if $a_{ij} = 0$:

$$\{\forall a_{ij} : a_{ij} > 0 \Leftrightarrow v(a_{ij}) > 0\} \wedge \{\forall a_{ij} : a_{ij} = 0 \Leftrightarrow v(a_{ij}) = 0\} \tag{2}$$

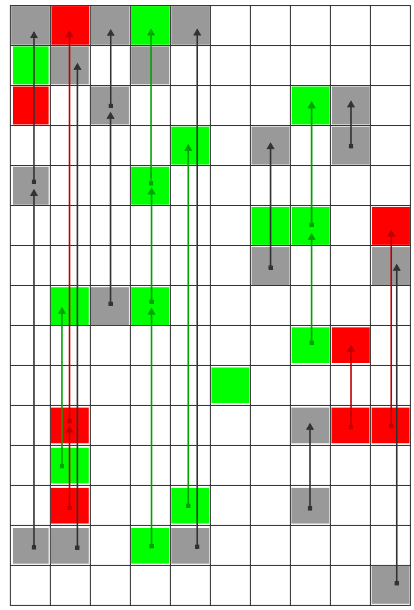

Figure 2. Vicinity transformation. The arrows indicate the position of the nearest same element in each column.

Reachability. If there exists a non-zero element a_{kj} in a column with lower index than k, then the corresponding b_{kj} element of vicinity matrix B is reachable from every non-zero element with higher indices than k by performing finite times iterated composition of vicinity transformation.

$$\{\exists k : 0 < k < i : a_{kj} = s_j\} \Rightarrow \{\forall i : i > k : a_{ij} = s_j \Leftrightarrow \exists c > 0 : v^c(a_{ij}) = b_{kj}\} \tag{3}$$

Termination. Iterated composition of vicinity transformation on any element a_{ij} is finite, i.e. it finally locates the first occurrence of 1 (i.e. the one with the lowest row index) in each column. Any further composition of vicinity transformation returns the same element.

$$\forall a_{ij} > 0 : \exists c > 0, k \geq 0 : \forall p \geq 0 : v_b^c(a_{ij}) = v_b^{c+p}(a_{kj}) = k \tag{4}$$

3.3 Propagation

Propagation is based on an observation that for a given set $A = \{a_1, a_2, \ldots a_n\}$ its power set $2^A = \{\emptyset, \{a_1\}, \{a_2\}, \ldots, \{a_n\}, \{a_1, a_2\}, \{a_{n-1}, a_n\}, \ldots, \{a_1, a_2, \ldots, a_n\}\}$ could be also denoted as the union of disjoint sets: $2^A = \emptyset \cup P_1 \cup P_2 \cup \ldots \cup P_n$, where $P_i \cup P_j = \emptyset$ and P_i is a family of sets that matches the following rule: the last element in each of its subsets is equal to a_i irrespective of the cardinality of the

set. For example, the third family of sets is equal to: $P_3 = \{\{a_3\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$.

Definition 6 (Motif propagation)**.** A family of sets is called a *propagation of motif* m and denoted as P_m if each of its subsets has its uttermost element equal to m.

Propagation of *seed* motif to any row r could be considered as simultaneously performing vicinity transformation on each column, until row r is reached in each column independently. Noteworthy, that based on Theorem 1, vicinity transformation will obtain intersection of *seed* and r.

Theorem 2. Every intersection of row a_i is a subset of P_i or is a subset of P_k, where $k > i$.

Proof. Suppose there existed a set S that contained an intersection of the element a_i and had uttermost element equal to a_k. If $k \leq i$, basing on Definition 6, a_i is the uttermost element of S and $k = i$, which is false. Hence, there must be $k = i$ or $k \geq i$ and basing on the Definition 6, there must be $S = P_i$ or $S = P_k$, where $k > i$. which ends the proof. □

Theorem 2 assures that all possible new inclusion-maximal biclusters containing a_i row will be generated by propagation of a_i motif (apart from intersections of a_i with rows with higher index than i). Restricting intersection that have already been found in rows with higher index and contain a_i will result in obtaining all inclusion-maximal biclusters, featuring row a_i.

3.4 Propagation-Based Biclustering Algorithm (PBBA)

The proposed solution called Propagation-Based Biclustering Algorithm (PBBA) adapts the main idea used in incremental algorithm version of Bimax, which bases on finding all inclusion-maximal cliques in graph. Visiting each node in Bimax finds all cliques containing the node, which involves iteration through all other nodes and extending localized biclusters to their maximality.

The main difference between PBBA and Bimax is the type of biclusters extracted from the input matrix. Bimax algorithm does not differentiate row patterns. Instead, it applies a threshold globally on the whole input matrix. All values above the threshold are considered equal. For instance, the values in two rows of the original input matrix may differ significantly, but if they are both above specified threshold, they will be assigned '1' by Bimax. Taking this into account, PBBA resembles xMotif with the difference, that PBBA aims at identifying all possible conserved motifs within the database, whereas xMotif does not.

The details of the algorithm have been presented in Algorithm 1 section.

The algorithm requires as input a matrix vicinity transformed according to Definition 5. By iterating through all rows of the input matrix (lines 4–16 of Algorithm 1), each one is regarded as a potentially new pattern. An array M_{all} of unordered maps M_i is used to store all retrieved biclusters. Combinations of columns

Algorithm 1 Propagation-based biclustering algorithm (PBBA)

```
1: procedure PBBA(matrix A)                          ▷ A – vicinity-transformed matrix
2:     M_all ← ∅                                     ▷ unordered map to store all biclusters
3:     R_all ← ∅                                     ▷ no restriction in each of restriction sets
4:     for i ← n ... 1 do                            ▷ set each row as seed
5:         M_i ← ∅                                   ▷ store all biclusters common with i^th pattern
6:         M_i ← insert(M_i, R_i, B(i, A_i*))        ▷ adding seed to retrieved biclusters
7:         mask ← A_i*    ▷ priority queue with all column transitions from i^th row
8:         {lev, pat} ← next_level(mask)
9:         while pat ≠ ∅ do                          ▷ proceed through all rows similar to seed
10:            M_i ← insert(M_i, R_i, B(lev ∪ i, pat))        ▷ intersect lev with M_i
11:            R_lev ← R_lev ∪ {pat}                 ▷ forbid addition of any subset of pat
12:            mask({j : mask(j) = lev}) ← v(mask(j))        ▷ proceed to next row
13:            {lev, pat} ← next_level(mask)
14:        end while
15:        M_all ← M_all ∪ M_i;
16:    end for
17:    print(M_all)
18: end procedure
```

that have already been indicated, as well as their subsets, are stored separately for each row in R_i unordered maps. Motif propagation, described in Definition 6, is implemented as a priority queue containing only active indices of columns, where vicinity transformation has not reached the first similar row to pattern.

Propagation starts from seeding the first pattern (n^{th} row), by calling *insert* function (line 6). Then a vicinity transformed matrix is used to find the next row *lev* with at least one element common with the i row. The next row, which has non-empty intersection with row i is determined by extracting top elements of priority queue within *next_level* function (lines 8 and 13). Propagation is continued (lines 9–14) until all rows with non-empty intersection with i row are visited.

```
1: function NEXT_LEVEL(mask)
2:     lev ← 0      ▷ index of next row similar to seed (top of mask priority queue)
3:     pat ← ∅                                       ▷ similarity between seed row and lev row
4:     lev ← max{j : v(mask(j)) ≠ j}     ▷ highest index of row is on top of queue
5:     pat ← {j : v(mask(j)) = lev}                  ▷ determine set of similar columns
6:     return lev, pat
7: end function
```

Expansion of biclusters containing row i is performed in *insert* function. A resulting set of biclusters similar to i, called M_i, is extended to its maximality by intersecting it with subsequent rows. This is achieved in each step by adding new sub-patterns to M_i or updating the existing ones with a new row.

```
1: function INSERT(unordered map M_i, unordered map R_i, bicluster B(x, y))
2:     if restricted(R_i, y) ∨ y = ∅ then
3:         return∅                          ▷ empty row or all subsets already identified
4:     end if
5:     if M_i[y] = ∅ then                   ▷ adding new bicluster: pattern or its intersection
6:         M_i[y] ← x
7:     else
8:         M_i[y] ← r ∪ x
9:     end if
10:    for all biclusters B(r, c) : M_i do
11:        col ← c ∩ y
12:        if ( ¬ restricted(R_i, col) ∧ col ≠ ∅) then
13:            if M_i[col] = ∅ then          ▷ find if col exists in unordered map
14:                M_i[col] ← x              ▷ subset of columns not found – adding
15:            else
16:                M_i[col] ← r ∪ x          ▷ subset of columns found – updating
17:            end if
18:        end if
19:    end for
20:    return M_i
21: end function
```

Prior to indicating any new bicluster, we need to check if each new pattern has not already been discovered in one of the earlier propagations. Otherwise, we might not achieve not maximal patterns. This verification is performed within *restricted* function (lines 2 and 12 of *insert* function).

```
1: function RESTRICTED(unordered map M_i, columns c)
2:     for all columns k : M_i do
3:         if k ⊇ c then                    ▷ new biclusters will not be maximal
4:             return TRUE
5:         end if
6:     end for
7:     return FALSE
8: end function
```

All restrictions for i^{th} row (i.e. intersection of i^{th} row with the rows that propagated to i) are stored within unordered map R_i. Before claiming any bicluster as pattern inclusion-maximal, its columns are compared with keys of R_i. Only those biclusters may be considered maximal, where propagation starts within i^{th} row. As the restriction set does not contain all of even 2^m subsets, which may be used as potential patterns, checking the restrictions requires verifying if the pattern is not a subset of any of pattern contained in R_i.

After considering intersections of seed row with *lev* row, the intersection pattern is considered as restricted for row *lev* in order to assure that non-maximal bicluster will not be generated when *lev* row becomes a seed (line 11 of Algorithm 1).

PBBA does not require discretized data as input. Values may be compared *per se* or be discretized into intervals (multi-level threshold). The algorithm does not aim at finding all constant bicluster from the matrix, but similar patterns appearing across columns. Achieving similar goal with Bimax algorithm would be quite challenging: initial discretization (same for the whole input matrix) would need to be performed separately for each row, as pattern uniqueness is an issue (that would increase Bimax complexity by N times). If any row contained the same value in column as the pattern, it would be given '1', otherwise '0'. After application of Bimax for each of N discretized datasets, an extra procedure should be executed that would filter out not maximal or repeating (i.e. with the same columns or same rows) biclusters.

3.5 Discussion on Algorithm Complexity

The main loop of the PBBA algorithm is executed n-times, where n is number of rows (genes). Inner loop of an algorithm, which may be executed up to n-times, includes execution of *next_level* and *insert* function.

Function *next_level*, which uses a priority queue of size m, involves pushing and popping and determining the maximal element. In each iteration, all elements may be popped and pushed back into queue (if rows are exactly the same). Access to the top element in the queue is considered constant, the popping element is up to twice logarithmic in size and the pushing element is considered amortized constant (if no reallocation happens, reallocation is linear in size). Therefore complexity of *next_level* is $\Theta(\log m)$ and in worst case $O(m)$.

The inner loop of *Insert* function (lines 10-19) is executed at most σ times, where σ is the number of pattern inclusion-maximal biclusters, associated with each of n patterns. It involves computing intersection (line 11) and union (line 16) of rows, which take $O(m)$ time, and calling *Restricted* function.

Restricted function verifies if a pattern found in a process of propagation has not been already discovered in earlier propagations. Restrictions are realized as an array of unordered map. Each unordered map stores intersection of the corresponding row with rows with higher indices. Finding if a subset does not belong to an unordered map may be accomplished in $\Theta(1)$ time. Detecting if a pattern is not a subset of a pattern requires iterating through all subsets of an unordered map. There may be at most n subsets, thus it may take up to $O(mn)$ to check column subsets of all succeeding rows, as for each pattern column comparison is necessary.

This gives $O(mn\sigma)$ for *Insert* function and final time complexity of $O(mn^3\sigma)$ for PBBA. This means that complexity of PBBA is not far from Bimax, which has $O(mn\beta \min\{m, n\})$ time complexity, where β is the number of all inclusion-maximal biclusters in a dataset [27]. Taking into account that PBBA stores the results of propagation separately in unordered maps we may assume $\beta \approx n\sigma$.

4 RESULTS

Our implementation of PBBA algorithm was written in C++ using Standard Template Library (STL – priority_queue) and Boost C++ Library (dynamic_bitset, unordered_map). Tests involved artificial datasets as well as real microarray datasets. For tests we used the authors' implementation in C++ of Bimax [27]. Implementation of xMotif written in C++ was taken directly from Biorithm-1.1 package [26]. We observed that Bimax algorithm is very sensitive to implementation. In many cases the results (number and sizes of biclusters) on 32-bits and 64-bits machines differed. Our implementation of Bimax for 32-bits and 64-bits worked slower, because of bit-tricks used in the original implementation, but the results were exactly the same. The algorithm xMotif required an extra input parameter, such as discriminating set of columns initial assignment ('c' option). We assigned all columns to one class and used default settings for further tests. Specifying different options for the xMotif algorithm resulted in Segmentation Fault errors, though the input was properly formatted.

4.1 Verification Test

In order to verify if the concept of algorithm is correct, a specific scenario was considered where the input dataset is binary. The tests included a comparison of the results yielded by PBBA and Bimax, xMotif algorithm was excluded from the tests. For binary datasets, PBBA and Bimax should yield exactly the same set of biclusters (possibly in a different order). The following datasets were used for tests:

- datasets in artificial scenario I and II from [27], including overlapping biclusters and noise – 182 datasets in total
- dataset with bio-synthetic pathways of *Arabidopsis thaliana* [33, 27]
- two datasets of yeast cell-cycle *Saccharomyces cerevisiae* [8, 29] included in BicAT-Plus package [1].

Artificial datasets from [27] involved cases of different sizes of input matrix and different noise levels. Within each matrix, ten different biclusters (overlapping or not) were planted. In this scenario, each of the artificial datasets was binarized with a threshold equal to the mean of the data, similarly to [27]. Real datasets were discretized within a threshold of 5 % in order to shorten computations. In each case, the results obtained by PBBA (after reordering) were exact to the Bimax results. This proved that PBBA algorithm is capable of finding inclusion-maximal biclusters.

The verification tests were performed in order to prove that PBBA algorithm for a specific (binary) dataset yields the same results as Bimax algorithm. Therefore Bimax algorithm may be considered as a special case of PBBA, in which the input data is discretized.

4.2 Binary Dataset Timings

For the running time analysis of the algorithms, we used the *Arabidopsis thaliana* dataset [33], provided by [27] and covered 734 rows and 69 columns. The dataset was discretized with consecutive thresholds. For each threshold, the test was repeated 5 times, the lowest and highest results were disregarded, others were averaged. The results of this test are presented in Table 4. For this test xMotif algorithm failed to discover any bicluster for every threshold. The average running times for three algorithms are presented in Table 1.

<table>
<tr><th></th><th>PBBA</th><th>Bimax</th><th>xMotif</th></tr>
<tr><td rowspan="2">Binary 5 %</td><td>0.305 s</td><td>0.034 s</td><td rowspan="10">xMotif failed
to detect
any bicluster
in discretized data.
On row data
it worked on
average 99.349 s
and found
77.0 biclusters.</td></tr>
<tr><td colspan="2">2 211 biclusters</td></tr>
<tr><td rowspan="2">Binary 10 %</td><td>2.125 s</td><td>0.257 s</td></tr>
<tr><td colspan="2">15 615 biclusters</td></tr>
<tr><td rowspan="2">Binary 15 %</td><td>17.615 s</td><td>1.788 s</td></tr>
<tr><td colspan="2">97 024 biclusters</td></tr>
<tr><td rowspan="2">Binary 20 %</td><td>122.528 s</td><td>10.905 s</td></tr>
<tr><td colspan="2">518 650 biclusters</td></tr>
<tr><td rowspan="2">Binary 25 %</td><td>741.137 s</td><td>64.771 s</td></tr>
<tr><td colspan="2">2 516 709 biclusters</td></tr>
</table>

Table 1. Running times of algorithms per discretization threshold

We conclude that xMotif algorithm is not suitable for binary datasets. PBBA and Bimax are very dependent on the level of binarization. The number of generated biclusters as well as running time increase where binarization level increases.

4.3 Input Parameters Evaluation

After confirming the PBBA's propriety, different tests were performed in order to determine algorithm's sensitivity for input data. As PBBA requires vicinity transformed matrix as input, for real datasets, the input matrix needs to be computed based on certain number of ranks. All values are sorted and assigned into specified number of ranks, providing approximately equal number of elements in each rank. Then, vicinity transformation is computed, according to Definition 5. The tests performed on Spellmen dataset included different number of ranks and different minimal sizes of the requested bicluster. The results are presented in Table 2. The tests show that input parameter (number of ranks) has huge impact on time of computations as well as the number of returned biclusters.

4.4 Biological Relevance

For testing biological significance, we used two well-established databases of *Saccharomyces cerevisiae*. The Gasch database [8] originally covered 6 152 genes under

Ranks	Minimal size	Number of Biclusters	Running Time
8 levels	14×14	100 779	14 h 51 m 58 s
8 levels	20×20	4	1 h 16 m 37 s
10 levels	16×16	2 221	30 m 1 s
12 levels	16×16	622	8 m 4 s
16 levels	10×10	41 766	14 m 54 s
16 levels	14×14	963	2 m 8 s
20 levels	12×12	1 899	1 m 25 s

Table 2. Number of biclusters of minimal size and running time of PBBA depending on number of ranks for Spellman dataset

173 different conditions, but it was narrowed to 2 993 genes by removing merged, deleted and retired genes by [1]. Similarly Spellman dataset was reduced from 6 778 down to 2 467 genes under 79 (of 82) conditions. The datasets are publicly available within the software BicAT-Plus from `http://www.bioinformatics.org/ftp/pub/bicat-plus/`. The original data was modified by background correction, normalization and missing value imputation by [1]. For calculating biological significance we used GeneMerge [3], which performs modified Bonferroni correction for over-representation. We used GO ontologies provided by [1] in BicAT-Plus package.

Before providing input for PBBA algorithm, data was ranked into an arbitrary number of steps, assuring that the input matrix has approximately the same number of elements of each value. Due to very long computation the minimal size of a bicluster needed to be set up. For Spellman dataset, PBBA was provided with data with 12 levels (minimal bicluster's size: 16×16) and 16 levels (12×12), and for Gasch with 25 levels (30×30).

The match score of overlapping biclusters was adapted from [27] and is similar to Jaccard index [10]: $J(G_1, G_2) = \frac{|G_1 \cap G_2|}{|G_1 \cup G_2|}$. It presents the number of genes common for both biclusters, divided by number of genes appearing in any of biclusters. We modified a standard procedure of postprocessing biclustering results used by [27, 17, 5] to filter out substantially overlapping biclusters. Instead of greedily choosing the largest bicluster (i.e. with the largest number of genes) non overlapping with previously selected more than 25 %, we firstly determined biological enrichment of each biclusters and then in increasing order of p-values we accepted those biclusters that do not overlap with others for more than 25 %.

We reckon that filtering out overlapping biclusters with p-value information provided beforehand is more objective and may take into account also those enriched biclusters, which could be disregarded in the standard procedure. The proposed filtering procedure asserts that most enriched biclusters are not lost before performing the comparison of algorithms.

In order to compare, in terms of biological relevance, after filtering procedure for each algorithm the 100 best biclusters were selected, similarly to [27]. Biclusters were considered enriched if any GO term was enriched with $p = 0.05$ level after performing

a modified Bonferroni correction based on the number of terms by GeneMerge [3]. Figure 3 shows the proportion of filtered enriched biclusters per the different level of significance.

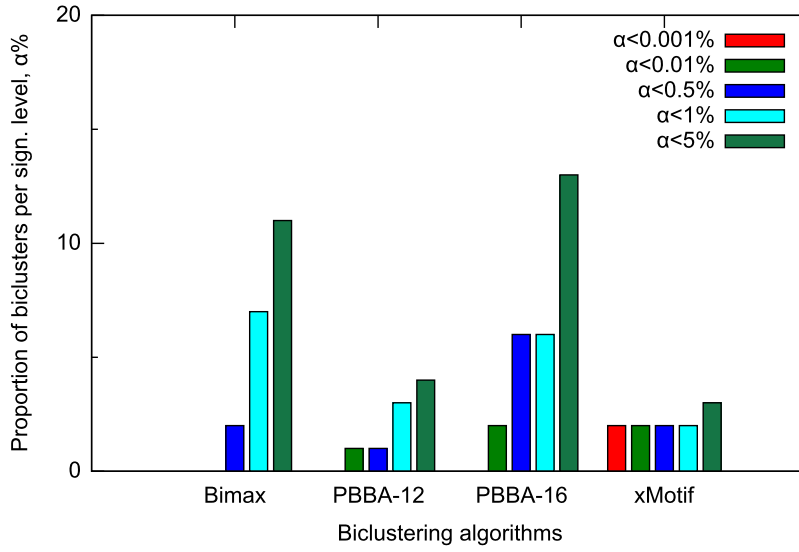

Figure 3. Proportion of biclusters significantly enriched by any GO Biological Process category for Spellman dataset

The most enriched GO terms for each of algorithms tested for Spellman dataset are presented in Table 3 and for Gasch dataset in Table 4.

The most enriched biclusters were found by xMotif algorithm, but only 3 managed to outlast a filtering procedure for overlapping biclusters in Spellman dataset. Filtering was sustained by 25 Bimax biclusters, 10 PBBA biclusters (with Spellman dataset discretized to 12 levels) and 27 PBBA biclusters (with data discretized to 16 levels). We conclude that xMotif algorithm returned mainly overlapping biclusters. The number of ranks for the PBBA algorithm has significant impact on the quality of the results, and the number and sizes of returned biclusters.

The running time of PBBA for the algorithms considering Spellman dataset are still within the level of acceptance. PBBA-12 on Intel Pentium M 2 GHz with 1.5 GB RAM needed 12 minutes 12 seconds to produce 622 biclusters (of at least 16×16 size) and PBBA-16 only 1 minute 10 seconds for 26 biclusters (16×16) and 2 minutes 57 seconds for 963 biclusters (14×14), respectively. In comparison, Bimax needed 6 minutes 32 seconds for computing 69 004 biclusters (min. size 16×16) and xMotif 46 minutes 18 seconds to return 23 biclusters (any size of biclusters accepted).

Name # of bicl. min. size	p-value	Functionally most enriched BP/MF/CC
Bimax	BP: 2.443e−3	GO:0006414: translational elongation
69 004	MF: 3.550e−4	GO:0004814: arginine-tRNA ligase activity
16×16	CC: 2.349e−3	GO:0005940: septin ring
PBBA12	BP: 3.793e−5	GO:0005981: regulation of glycogen catabolic process
622	MF: 4.636e−3	GO:0016538: cyclin-dep. protein kinase reg. activity
16×16	CC: 6.020e−4	GO:0000307: cycl.-dep. prot. kinase holoenzyme compl.
PBBA16	BP: 1.335e−5	GO:0005981: regulation of glycogen catabolic process
963	MF: 5.365e−4	GO:0003843: 1,3-beta-glucan synthase activity
14×14	CC: 3.073e−4	GO:0000307: cycl.-dep. prot. kinase holoenzyme compl.
xMotif	BP: 2.927e−10	GO:0006412: translation
23	MF: 2.512e−13	GO:0003735: structural constituent of ribosome
all	CC: 2.629e−13	GO:0005842: cytosolic large ribosomal subunit

Table 3. The most enriched GO term: Biological Process (BP), Molecular Function (MF) and Cellular Component (CC) found by algorithms for Spellman dataset

Name bicl. #	p-value	Functionally most enriched BP/MF/CC
Bimax	BP: 5.339e−3	GO:0006281: DNA repair
375 935	MF: 1.475e−3	GO:0003704: specific RNA polymerase II TF activity
	CC: 1.574e−2	GO:0000228: nuclear chromosome
PBBA	BP: 1.481e−4	GO:0005981: regulation of glycogen catabolic process
69004	MF: 1.378e−3	GO:0004596: peptide alpha-N-acetyltransferase activity
	CC: 2.054e−3	GO:0000307: cyclin-dep. prot. kinase holoenzyme complex
xMotif	BP: 1.164e−15	GO:0042254: ribosome biogenesis and assembly
19	MF: 1.850e−9	GO:0003735: structural constituent of ribosome
	CC: 5.146e−14	GO:0005842: cytosolic large ribosomal subunit

Table 4. The most enriched GO term: Biological Process (BP), Molecular Function (MF) and Cellular Component (CC) found by algorithms for Gasch dataset

The number of input ranks for the algorithm as well as the minimal size of the bicluster play a significant role in the time of computation. Overbalanced input parameters may end up in excessive time of computations and producing millions of biclusters, similarly to Bimax. We reckon that the proper specification of input parameters enables PBBA to easily compete in terms of running times against Bimax, which is considered as a very fast biclustering approach [5].

5 SUMMARY

In this article, we propose theoretical background of vicinity transformation for detecting column similarities within the input matrix. The proposed PBBA al-

gorithm, which uses vicinity transformed matrix, is able to compute all pattern inclusion-maximal biclusters in the acceptable time.

We reckon that Bimax may be considered as a special case of PBBA for discretized input. Preparing data for PBBA may be considered as performing multilevel threshold for real datasets. We proved that PBBA algorithm may easily extend Bimax in terms of the flexibility as well as the quality of results. The proper setting of PBBA input parameters makes PBBA challenge other algorithms also in the term of speed.

PBBA, similarly to Bimax, may return a large number (hundreds of thousands or even millions) of partially overlapping biclusters. Special procedure needs to be performed in order to filter unique results. Computations may exceed even to hours on Intel Xeon 2.4 GHz processor. Therefore a special caution needs to be taken when determining the number of levels of the input data for PBBA algorithm, as well as the size of minimal bicluster. Increasing the number of levels usually increases the algorithm's sensitivity, but causes a larger dispersion of biclusters and reduces the sizes of returned biclusters.

The proposed algorithm may be successfully applied to biological data potential and flexibility. PBBA may be regarded as a hybrid combining speed of Bimax and the concept of conserved gene expression motif of xMotif. Further tests are required to prove PBBA usability on different biological datasets.

Further research will include verification if a pruning procedure for propagations may be applied in order to reduce running times of the algorithm. A separate study on elimination of multiple overlaps should be performed. Rough biclustering [19] may be used as a good starting point.

Consecutive propagations may use the same pattern, therefore there is a potential to halt propagation prematurely before reaching the first similar row. Tests on other datasets, for example on protein databases [22] and comparison with other biclustering algorithms, for example eBi [30] or HRoBi [20], may also determine the exact application areas of the algorithm.

Separate tests need to be performed for adjusting values of the input parameters. Determining number of ranks in input matrix and including data-specific issues, such as opposite effect of up/down-regulated genes, may also improve algorithm's accuracy.

Acknowledgements

This research was funded by the Polish National Science Center (NCN), grant No. 2013/11/N/ST6/03204. This research was supported in part by PL-Grid Infrastructure. The authors would like to express their sincere appreciation to prof. Jacek Kitowski for his valuable remarks and to the Academic Computer Centre CYFRONET AGH for providing computing power within the POWIEW project.

REFERENCES

[1] Alakwaa, F.—Solouma, N.—Kadah, Y.: Construction of Gene Regulatory Networks Using Biclustering and Bayesian Networks. Theoretical Biology and Medical Modelling, Vol. 8, 2011, No. 1, Art. No. 39.

[2] Ben-Dor, A.—Chor, B.—Karp, R.—Yakhini, Z.: Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem. Journal of Computational Biology, Vol. 10, 2003, No. 3-4, pp. 373–384.

[3] Castillo-Davis, C. I.—Hartl, D. L.: GeneMerge – Post-Genomic Analysis. Data Mining, and Hypothesis Testing. Bioinformatics, Vol. 19, 2003, No. 7, pp. 891–892.

[4] Cheng, Y.—Church, G.: Biclustering of Expression Data. Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology. Vol. 8, 2000, pp. 93–103.

[5] Eren, K.—Deveci, M.—Küçüktunç, O.—Çatalyürek, Ü.: A Comparative Analysis of Biclustering Algorithms for Gene Expression Data. Briefings in Bioinformatics, Vol. 14, 2013, No. 3, pp. 93–103.

[6] Erten, C.—Özdinler, M.: Biclustering Expression Data Based on Expanding Localized Substructures. Bioinformatics and Computational Biology, Lecture Notes in Computer Science, Vol. 5462, 2009, pp. 224–235.

[7] Ford, L. R.—Fulkerson, D. R.: Maximal Flow Through a Network. Canadian Journal of Mathematics, Vol. 8, 1956, No. 3, pp. 399–404.

[8] Gasch, A.—Spellman, P.—Kao, C.—Carmel-Harel, O.—Eisen, M.—Storz, G.—Botstein, D.—Brown, P.: Genomic Expression Programs in the Response of Yeast Cells to Environmental Changes. Science Signalling, Vol. 11, 2000, No. 12, pp. 4241–4257.

[9] Gu, J.—Liu, J.: Bayesian Biclustering of Gene Expression Data. BMC Genomics, Vol. 9, 2008, Suppl. 1, S4, 10 pp. doi: 10.1186/1471-2164-9-S1-S4.

[10] Halkidi, M.—Batistakis, Y.—Vazirgiannis, M.: On Clustering Validation Techniques. Journal of Intelligent Information Systems, Vol. 17, 2001, No. 2, pp. 107–145.

[11] Hochreiter, S.—Bodenhofer, U.—Heusel, M.—Mayr, A.—Mitterecker, A.—Kasim, A.—Khamiakova, T.—Van Sanden, S.—Lin, D.—Talloen, W. et al.: FABIA: Factor Analysis for Bicluster Acquisition. Bioinformatics, Vol. 26, 2010, No. 12, pp. 1520–1527.

[12] Horzyk, A.: Information Freedom and Associative Artificial Intelligence. Artificial Intelligence and Soft Computing, Springer Verlag Berlin Heidelberg, Lecture Notes in Artificial Intelligence, Vol. 7267, 2012, pp. 81–89. ISBN 978-3-642-29346-7.

[13] Horzyk, A.: How Does Human-Like Knowledge Come into Being in Artificial Associative Systems. Proceedings of the 8th International Conference on Knowledge, Information and Creativity Support Systems, Krakow, Poland, 2013, pp. 189–200.

[14] Ihmels, J.—Bergmann, S.—Barkai, N.: Defining Transcription Modules Using Large-Scale Gene Expression Data. Bioinformatics, Vol. 20, 2004, No. 13, pp. 1993–2003.

[15] Kluger, Y.—Basri, R.—Chang, J. T.—Gerstein, M.: Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. Genome Research, Vol. 13, 2003, No. 4, pp. 703–716.

[16] Lazzeroni, L.—Owen, A.: Plaid Models for Gene Expression Data. Statistica Sinica, Vol. 12, 2002, No. 1, pp. 61–86.

[17] Li, G.—Ma, Q.—Tang, H.—Paterson, A.—Xu, Y.: QUBIC: A Qualitative Biclustering Algorithm for Analyses of Gene Expression Data. Nucleic Acids Research, Vol. 37, 2009, No. 15, Art. No. e101, 10 pp. doi: 10.1093/nar/gkp491.

[18] Madeira, S.—Oliveira, A.: Biclustering Algorithms for Biological Data Analysis: A Survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), Vol. 1, 2004, No. 1, pp. 24–45.

[19] Michalak, M.: Foundations of Rough Biclustering. Artificial Intelligence and Soft Computing, Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7268, 2012, pp. 144–151.

[20] Michalak, M.—Stawarz, M.: HRoBi – The Algorithm for Hierarchical Rough Biclustering. Artificial Intelligence and Soft Computing, Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7895, 2013, pp. 194–205.

[21] Murali, T.—Kasif, S.: Extracting Conserved Gene Expression Motifs from Gene Expression Data. Proceedings of Pacific Symposium on Biocomputing. Vol. 8, 2003, pp. 77–88.

[22] Orzechowski, P.—Boryczko, K.: Parallel Approach for Visual Clustering of Protein Databases. Computing and Informatics, Vol. 29, No. 6+, 2010, pp. 1221–1231.

[23] Orzechowski, P.: Proximity Measures and Results Validation in Biclustering – A Survey. Artificial Intelligence and Soft Computing, Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7895, 2013, pp. 206–217.

[24] Orzechowski, P.—Boryczko, K.: Hybrid Biclustering Algorithms for Data Mining. Applications of Evolutionary Computation, 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30–April 1, 2016, Proceedings, Part I, 2016, pp. 156–168, ISBN 978-3-319-31204-0.

[25] Orzechowski, P.—Boryczko, K.: Text Mining with Hybrid Biclustering Algorithms. Artificial Intelligence and Soft Computing, 15th International Conference, ICAISC 2016, Zakopane, Poland, June 12-16, 2016, Proceedings, Part II, Springer International Publishing, 2016, pp. 102–113. ISBN 978-3-319-39384-1.

[26] Poirel, C. L.—Rahman, A.—Rodrigues, R. R.—Krishnan, A.—Addesa, J. R.—Murali, T. M.: Reconciling Differential Gene Expression Data with Molecular Interaction Networks. Bioinformatics, Vol. 29, 2013, No. 5, pp. 622–629.

[27] Prelić, A.—Bleuler, S.—Zimmermann, P.—Wille, A.—Bühlmann, P.—Gruissem, W.—Hennig, L.—Thiele, L.—Zitzler, E.: A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data. Bioinformatics, Vol. 22, 2006, No. 9, pp. 1122–1129.

[28] Schrijver, A.: On the History of the Transportation and Maximum Flow Problems. Mathematical Programming, Vol. 93, 2002, No. 3, pp. 437–445.

[29] Spellman, P.—Sherlock, G.—Zhang, M.—Iyer, V.—Anders, K.—Eisen, M.—Brown, P.—Botstein, D.—Futcher, B.: Comprehensive Identifica-

tion of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization. Molecular Biology of the Cell, Vol. 9, 1998, No. 12, pp. 3273–3297.

[30] Stawarz, M.—Michalak, M.: eBi – The Algorithm for Exact Biclustering. Artificial Intelligence and Soft Computing, Springer Berlin Heidelberg, Lecture Notes in Computer Science, Vol. 7268, 2012, pp. 327–334.

[31] Tanay, A.—Sharan, R.—Shamir, R.: Discovering Statistically Significant Biclusters in Gene Expression Data. Bioinformatics, Vol. 18, 2002, Suppl. 1, pp. S136–S144.

[32] Topa, P.—Młocek, P.: Using Shared Memory as a Cache in Cellular Automata Water Flow Simulations on GPUs. Computer Science, Vol. 14, 2013, No. 3, pp. 385–401.

[33] Wille, A.—Zimmermann, P.—Vranová, E.—Fürholz, A.—Laule, O.—Bleuler, S.—Hennig, L.—Prelic, A.—Von Rohr, P.—Thiele, L.—Zitzler, E.—Gruissem, W.—Bühlmann, P.: Sparse Graphical Gaussian Modeling of the Isoprenoid Gene Network in Arabidopsis Thaliana. Genome Biology, Vol. 5, 2004, No. 11, Art. No. R92.

Patryk Orzechowski received his Ph.D. degree in computer science in 2015 from AGH University of Science and Technology, Krakow, Poland, where he works as a research-and-teaching assistant. His scientific interests include bioinformatics, artificial intelligence and data mining. He is also interested in mobile technologies and big data.

Krzysztof Boryczko received his Ph.D. degree in computer science in 1996 from the Department of Computer Science, AGH University of Science and Technology, Krakow, Poland, where he is now Associate Professor. His research interests focus on large scale simulations with particle methods. He is also interested in feature extraction, clustering methods for analysis of simulation data and scientific visualization.