BiSim: A Simple and Efficient Biclustering Algorithm

Nighat Noureen, Muhammad Abdul Qadir

Department of Computer Science & Bioinformatics, Mohammad Ali Jinnah University Islamabad Campus
Mohammad Ali Jinnah University, Jinnah Avenue, Blue Area Islamabad, Pakistan

nighatnoureen@gmail.com, aqadir@jinnah.edu.pk

Abstract— **Analysis of gene expression data includes classification of the data into groups and subgroups based on similar expression patterns. Standard clustering methods for the analysis of gene expression data only identifies the global models while missing the local expression patterns. In order to identify the missed patterns biclustering approach has been introduced. Various biclustering algorithms have been proposed by scientists. Among them Binary Inclusion maximal algorithm (BiMax) forms biclusters when applied on a gene expression data through divide and conquer approach. The worst-case running-time complexity of BiMax for matrices containing disjoint biclusters is O(nmb) and for arbitrary matrices is of order O(nmb min{n, m}) where b is the number of all inclusion-maximal biclusters in matrix. In this paper we present an improved algorithm, BiSim, for biclustering which is easy and avoids complex computations as in BiMax. The complexity of our approach is O(n*m) for n genes and m conditions, i.e, a matrix of size n*m. Also it avoids extra computations within the same complexity class and avoids missing of any biclusters.**

Keywords— ***Biclustering, gene expression data, biclustering algorithm, bicluster, co-regulated, co-expressed, local expression patterns***

I. INTRODUCTION

Gene expression plays an important role in cell differentiation, development, and pathological behaviour [1].The remarkable ability of monitoring simultaneously the expression levels of thousands of genes in a cell has been provided by the DNA microarrays to the biologists [2], [3]. Gene expression analyses using high throughput techniques provides insights into cell function and also stimulate the development of new therapies and diagnostics [1].

Matrices are used to organize the gene expression data where rows in a matrix represent genes, columns represent various samples such as tissues or experimental conditions, and numbers in each cell characterize the expression level of the particular gene in the sample. Matrices of gene expression have been extensively analysed in two dimensions: the gene dimension and the condition dimension [4]. By comparing rows in the matrix expression patterns of genes are analysed and by comparing columns in the matrix expression patterns of samples are analysed.

Clustering identifies similarly expressed groups of genes including all the experimental conditions as in Fig 1, while biclustering identifies the subgroups of genes based on subgroups of conditions as in Fig 2. So a bicluster has the ability to identify the patterns which are normally missed by clustering. In other words the bicluster has the ability to identify those biological processes which cover few conditions.

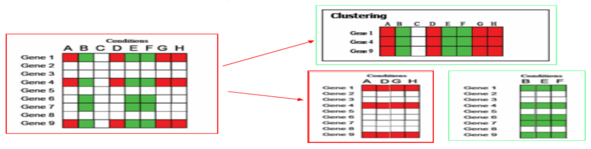


Fig. 1 Example of clusters obtained from a gene expression matrix

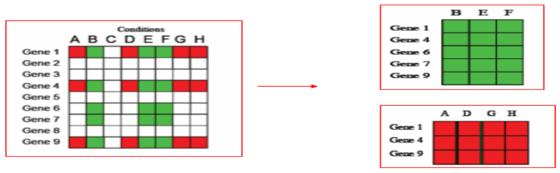


Fig. 2 Example of biclusters obtained from a gene expression method

Clustering is being considered as the main approach for analysis of gene expression data [5]. In many contexts Clustering methods have proved successful [5]. Initially hierarchical clustering techniques were used for the analysis of gene expression data [6]. Clustering methods have been successfully applied in one of the studies given in [7].

One of the major drawbacks of commonly used clustering algorithms is that they assign each gene to a single cluster, while in fact genes can be included in multiple clusters because of the fact that they participate in several functions [8], [9], [10]. Identifying patterns that are common to only a part of the expression data matrix in clustering methods is also difficult [5]. In standard clustering methods, genes are analysed under all experimental conditions based on their expression. Usually cellular processes are affected only by a small subset of conditions so it is problematic [11]. In Clustering genes are partitioned into disjoint sets such that it implies the association of each gene with a single biological process, which may be an oversimplification of the biological system [12]. In cellular processes under certain experimental conditions subsets of genes are co-regulated and co-expressed, but they behave almost independently under other conditions [5]. Many genetic pathways that are not apparent otherwise can be discovered by uncovering the local expression patterns [5].

To develop algorithm capable of discovering local patterns in microarray data is therefore highly desirable rather than sticking to the clustering paradigm [5]. Hartigan [14] in 1975 introduced biclustering. In order to identify local patterns in gene expression data several biclustering methods have been suggested. A *bicluster* is defined as a subset of genes that exhibit compatible expression patterns over a subset of conditions [13].

This paper presents a new and simple method of finding biclusters in a gene expression method. A brief review of existing biclustering approaches has also been presented.

II. BICLUSTERING APPROACHES

Various biclustering methods proposed in literature follow one of the following approaches according to [4]:

1. Divide and Conquer
2. Greedy Search
3. Iterative row and column clustering combination
4. Distribution parameter identification.
5. Exhaustive bicluster enumeration.

Here we discuss briefly the strengths and weakness of each approach separately.

A. Divide and Conquer

This approach breaks the given problem into various sub problems which are similar to the original problem but only the size is reduced. The sub problems are solved recursively and then solutions are returned. The solutions to the sub problems are combined to produce the solution to the original problem [16].

The significant advantage of Divide-and-conquer algorithms is that they are potentially very fast. Along with their significant speed a very significant drawback of this approach is that it is very likely to miss good biclusters that may be split before they can be identified [4]. As the division is done based upon the values in the cells, therefore, the matrix has to be traversed many times in order to indentify the cells for each divided portion. This will increase the computation even within the same complexity class.

Among the biclustering algorithms which follow divide and conquer approach are Block clustering and BiMax [13]. Block clustering is a top down, row and column clustering of the data matrix and is the first algorithm to follow this approach [4].

B. Greedy Search.

Greedy search methods choose a solution which seems to be optimal locally. It is then considered that this will give a globally optimal solution [16]. As far as the weakness of this approach is concerned it may make wrong decisions to consider due to its greedy nature and may loose good biclusters. Potentially they seem to be fast. The very first biclustering method applied to gene expression data [8] follows the greedy approach. Along with the Cheng and Church method the Order Preserving Submatrix Algorithm, *OPSM* [5]; and xMotif [1] also work according to the same approach.

C. Iterative Row and Column Clustering Combination.

Using this approach clustering algorithms are applied to rows and columns of the data matrix separately in order to identify biclusters and then using some sort of iterative procedure the results of the two cluster arrangements are combined [4].

Several algorithms use this iterative row and column clustering combination idea. Among them, The Coupled Two-Way Clustering (CTWC) [10], The Interrelated Two-Way Clustering (ITWC) [17] and The Double Conjugated Clustering (DCC) [18] work according to this approach. In CTWC based on iterative row and column clustering combination when subsets of rows or columns are identified as stable clusters in clustering iterations then only they are the candidates for the next iteration[4].This shows a drawback of missing the significant biclusters by just only passing the produced clusters in next iteration.

D. Distribution Parameter Identification.

The biclusters are generated using a given statistical model and the distribution parameters that fit, in the best way, the available data, by minimizing a certain criterion through an iterative approach are identified [4]. Plaid model [19] follows this approach.

E. Exhaustive Bicluster Enumeration.

By assuming restrictions on the size of the biclusters that should be listed, a number of methods have been used to speed up exhaustive search.

These algorithms certainly find the best biclusters, if they exist, but have a very serious drawback. They can only be executed by assuming restrictions on the size of the biclusters due to their high complexity. Assuming some restrictions on the input instances and using efficient algorithmic techniques designed to make the enumeration feasible; these methods perform exhaustive enumeration [4].

Statistical-Algorithmic Method for Bicluster Analysis (SAMBA) [12] follows this approach.

Existing methods and biclustering approaches though have strengths and weaknesses but still they produce good results. As research continues, new ideas come into minds of the researches which further lead to the origination of new methods and approaches. The basic idea behind these newly proposed methods is either to propose a totally novel way for solving a particular problem or based on some previous method, proposing a better way instead of the existing approach. Based on the idea of improving the existing approach a new method BiSim has been presented. It works on the binary matrix of gene expression data like the existing method BiMax [13]. BiMax follows divide and conquer strategy but BiSim instead of this follows an iterative approach of identifying the biclusters. It avoids the extra matrix traversals which in other way cannot be avoided in case of BiMax due to recursion.

III. BICLUSTERING CLASSES

Biclusters formed by the biclustering algorithms belong to one of the four major classes:

1. Bicluster with constant values.
2. Bicluster with constant values on rows or columns.
3. Bicluster with coherent values.
4. Bicluster with coherent evolutions.

In the first three classes the numeric values in the data matrix are directly analyzed and subsets of rows and subsets of columns with similar behaviors are searched [4]. Regardless of the exact numeric values in the data matrix the fourth class finds the coherent behaviors [4]. The elements in the datamatrix are view as symbols in biclusters with coherent evolutions [4].Among the biclustering methods the BiMax algorithm [13] lies in the class of constant values Bicluster. The CC [8] algorithm lies in Bicluster with coherent values, and xMotifs [1], SAMBA [12] and OPSM [5] form Bicluster with coherent evolutions.

IV. Binary Inclusion Maixmal Algorithm (BiMax)

The biclustering algorithm BiMax [13] takes the two dimensional matrix as input having zero's and ones as the entries of the matrix. One's represent that a particular gene expresses itself in the respective condition while zero means that there is no expression. The BiMax algorithm follows the Divide and Conquer approach in order to identify the biclusters.

A set of ***m*** microarray experiments for ***n*** genes can be represented by a binary matrix $\boldsymbol{E_{n\times m}}$, where a cell $\boldsymbol{e_{ij}}$ is 1 whenever gene ***i*** responds in the condition ***j*** and otherwise it is 0 [13]. A *bicluster* (G,C) corresponds to a subset of genes that jointly respond across a subset of samples. Therefore the pair (G,C) defines a submatrix of ***E*** for which all elements equal to 1 [13]. The idea of the Bimax algorithm is to partition the input matrix say ***E*** into three submatrices. One of them contains only 0-cells and therefore can be excluded in the following processing. The remaining two submatrices are called as ***U*** and ***V*** by the authors. The BiMax algorithm is then recursively applied to ***U*** and ***V***; the recursion ends if the current matrix represents a bicluster, i.e., contains only 1s. If ***U*** and ***V*** do not share any rows and columns of ***E***, i.e., ***GW*** are empty, the two matrices can be processed independently from each other. However, if ***U*** and ***V*** have a set ***GW*** of rows in common, special care is necessary to only generate those biclusters in ***V*** that share at least one common column with ***CV***. The idea of BiMax algorithm presented by Prelic et al is illustrated by us in an example in Fig 3, Fig 4 and Fig 5. These figures will give an idea how BiMax works.

1	1	0	0	0
1	1	0	0	0
1	1	1	0	0
0	0	0	1	1
0	0	0	1	1

Fig. 3 Input matrix E for BiMax algorithm

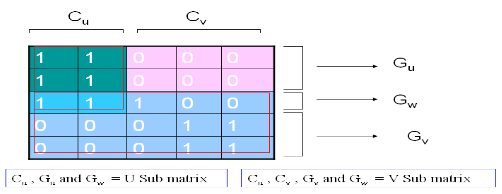

C_u
C_v
G_u
G_w
G_v
C_u, G_u and G_w = U Sub matrix
C_u, C_v, G_v and G_w = V Sub matrix

Fig. 4 Sub matrices U and V formation in input matrix E by BiMax algorithm

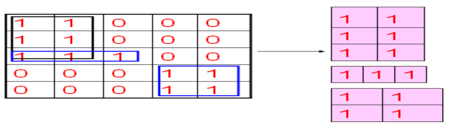


Fig. 5 Biclusters formed from input matrix E by BiMax algorithm

V. Problem Identified

As mentioned that BiMax works on the divide and conquer approach, it divides the input matrix into sub matrices and then recursively apply the algorithm on it. Recursion as we know needs a lot of computations and what if the input matrix is such that it is very large and the BiMax after applied on it is unable to find any bicluster? In this way the use of this recursive approach would not be efficient. Division of matrix into sub matrices require the traversal of the complete matrix and these traversals increase with the subsequent recursive calls. In order to avoid these overheads of recursive approach of finding biclusters we propose an easy and efficient iterative approach of finding biclusters from the input binary matrix which is meaningful and avoid complex computations.

VI. Proposed Approach

Our approach in BiSim traverses the matrix and saves the column starting index, column ending index and row number for each row wise bicluster. Identification of row wise bicluster involves the working of two major outer loops. The outer for loop is for the manipulation of rows and the while loop inner to for loop is for the manipulation of columns of the matrix. This traversal of the matrix and the saving of the row wise biclusters the complexity makes the complexity class O (m*n). After having row wise biclusters, two types of comparisons are performed. First comparison is column index based and second is row index based. In column index based comparison the column start and column end of row wise biclusters are compared and on the basis of similarities they are combined to get larger sets (biclusters). Similarly in row index based the columns expressing in similar rows are combined. For these indices manipulation there will be no need to traverse the matrix again. The column indexed based comparison comprising of nine cases and row-indexed based comparison work in two for loops; one of the manipulation of rows and the second for the columns. Altogether these computations make the complexity class of the algorithm as O (n*m) with only one traversal of the matrix as mentioned in row wise bicluster identification part of the pseudo code.

The algorithm with all its cases and chunks of pseudo code is explained as follows:-

A. Identification of Row-Wise Biclusters:

The row wise biclusters are stored in a vector. Each location of a vector represents a bicluster having the information of column start index, column end index and row number. Figure 6 represents the vector having the row biclusters.

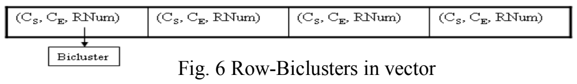


Fig. 6 Row-Biclusters in vector

The Pseudo code for obtaining row-wise biclusters is as follows:

```
for each row i
{
   While (j<=ColSize)
   {
      while (Array[i][j] ==0 AND j <= ColSize)
           ++j;
      If ( j < ColSize)
      {
         RowNum[index]= i
         ColStart = j;
         While( Array [i][j] = = 1 && j<=ColSize)
              j++;
       ColEnd=j;
       Bicluster(ColStart, ColEnd, RowNum)
       Vector. push (Bicluster)
      }
   }
}
```

After running this pseudo code the input matrix will form row biclusters and we will have a Vector of biclusters. Bicluster in the pseudo code is an entity containing column start index represented as ColStart, column end index represented as ColEnd and the row numbers of corresponding biclusters as a one dimensional array called as RowNum in the pseudo code.

B. Combining Clusters:

The row-wise biclusters are combined as described above on the basis of similarities. It includes various cases which are explained in detail with the help of pseudo code and diagrams. In each case ColStart represents the column starting index of a bicluster, ColEnd represents column ending index and RowNum is a one dimensional array which contains the row numbers of bicluster. Vector is the list containing biclusters. Vector[i] is the current index position and Vector [i+1] is the next to current index position. The NewBicluster is an entity like the Bicluster in above section.

1. Case 1:

In this case two similar row biclusters are combined to form one larger and the original two are fully included in the new bicluster so they will be removed and new one will be saved in vector. This is explained in pseudo code and Fig.7 below. The result of this case is shown in figure 17 of test cases to test this algorithm.

Pseudo code Case 1:

```
if((ColStart of Vector[i] == ColStart of Vector[i+1] && (ColEnd of
Vector[i] ==ColEnd of Vector[i+1]))
{
    NewBicluster(ColStart,ColEnd,,RowNum[i], RowNum[i+1]);
    Vector. Push(NewBicluster);
    Vector.delete (Vector[i]);
    Vector.delete(Vector[i+1]);
}
```

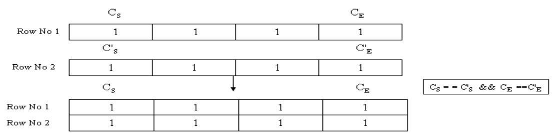


Fig. 7 CS = = C'S && CE ==C'E

2. Case 2:

Here column start index and column end index of both clusters are different but the similar columns are picked and combined. Pseudo code below and Fig.8 explains the case. First two rows of the cluster in figure 23 for our test cases is build by this part of the algorithm.

Pseudo code Case 2:

```
if((ColStart of Vector[i] < ColStart of Vector[i+1] && (ColEnd of
Vector[i] >ColEnd of Vector[i]+1))

{NewBicluster(ColStart[i+1],ColEnd[i+1],RowNum[i],RowNum[i
+1]);
    Vector. Push(NewBicluster);
}
```

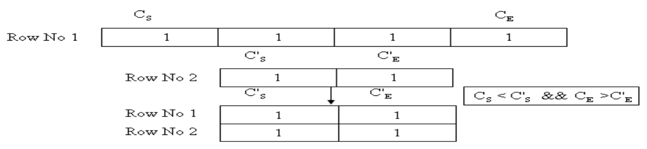


Fig. 8 CS < C'S && CE >C'E

3. Case 3:

Like Case 2 the column indices here are also different, so only similar ones are combined. Pseudo code and Fig.9 explains the case. Row 2& 3 of our test case in fig 23 is build by this part of algorithm.

Pseudo code Case 3:

```
if((ColStart of Vector[i] > ColStart of Vector[i+1] && (ColEnd
of Vector[i] <ColEnd of Vector[i+1]))

{NewBicluster(ColStart[i],ColEnd[i],RowNum[i],
RowNum[i+1]);
    Vector. Push(NewBicluster);
}
```

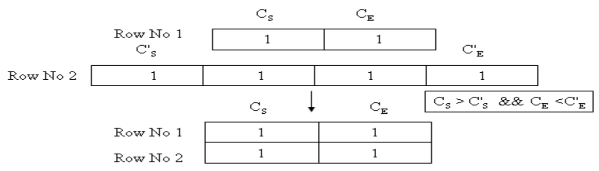


Fig. 9 CS > C'S && CE <C'E

4. Case 4:

Here the column start indices are same but column end indices are different, so only similar ones are combined. Figure 10 and the pseudo code below explain the case.

Pseudo code Case 4:

```
if((ColStart of Vector[i] == ColStart of Vector[i+1] && (ColEnd of
Vector[i] <ColEnd of Vector[i+1]))

{ NewBicluster(ColStart[i],ColEnd[i],RowNum[i],RowNum[i+1]);
    Vector. Push(NewBicluster);
}
```

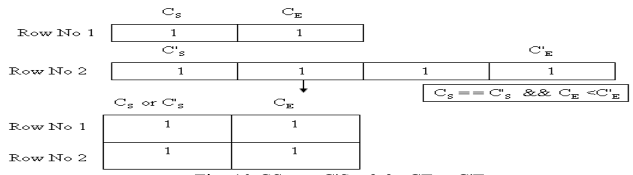


Fig. 10 CS = = C'S && CE <C'E

5. Case 5:

Case 5 is the similar as case 4. The only difference is the exchange of values in both. Figure 11 and the pseudo code below explain the case.

Pseudo code Case 5:

```
if((ColStart of Vector[i] == ColStart of Vector[i+1] && (ColEnd of
Vector[i] >ColEnd of Vector[i+1]))

{ NewBicluster(ColStart[i],ColEnd[i+1],RowNum[i],RowNum [i+1]);
    Vector. Push(NewBicluster);
}
```

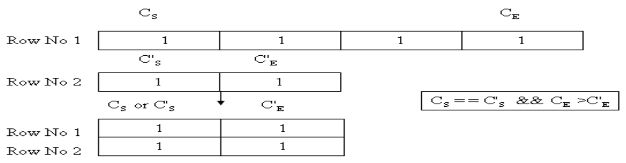


Fig. 11 CS = = C'S && CE >C'E

6. Case 6:

In case 6 unlike the column start the column end indices are same and the column start indices are different. Figure 12 and the pseudo code below explain the case.

Pseudo code Case 6:

```
if((ColStart of Vector[i] <ColStart of Vector[i+1] && (ColEnd of Vector[i] ==ColEnd of Vector[i+1]))

{NewBicluster(ColStart[i+1],ColEnd[i],RowNum[i],RowNum [i+1]);
        Vector. Push(NewBicluster);
}
```

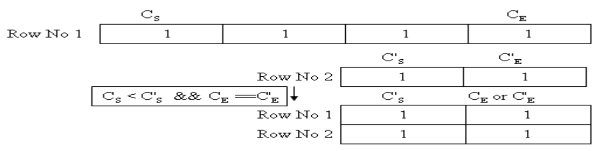


Fig. 12 CS < C'S && CE ==C'E

7. Case 7:

Case 7 is same as case 6 in which the column end indices are same. Figure 13 and the pseudo code below explain the case.

Pseudo code Case 7:

```
if((ColStart of Vector[i] >ColStart of Vector[i+1] && (ColEnd of Vector[i] ==ColEnd of Vector[i+1]))
{
  NewBicluster(ColStart[i],ColEnd[i],RowNum[i], RowNum [i+1]);
 Vector. Push(NewBicluster);
}
```

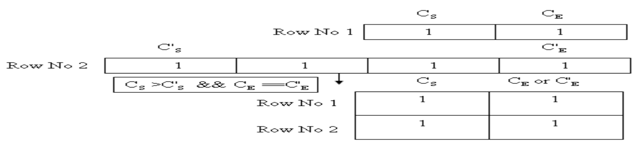


Fig. 13 CS >C'S && CE ==C'E

8. Case 8:

In this case column start and end index of first bicluster are lesser than the second one. The similar column indices make a bicluster. Figure 14 and the pseudo code below explain the case.

Pseudo code Case 8:

```
if((ColStart of Vector[i] <ColStart of Vector[i+1] && (ColEnd of Vector[i] <ColEnd of Vector[i+1]))

{ NewBicluster(ColStart[i+1],ColEnd[i],RowNum[i],RowNum [i+1]);
   Vector. Push(NewBicluster);
}
```

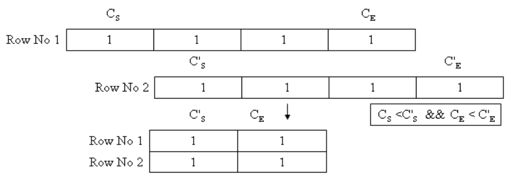


Fig. 14 CS <C'S && CE < C'E

9. Case 9:

In this case column start and end index of first bicluster are greater than the second one. The similar column indices make a bicluster. Figure 15 and the pseudo code below explain the case.

Pseudo code Case 9:

```
if((ColStart of Vector[i] >ColStart of Vector[i+1] && (ColEnd of Vector[i] >ColEnd of Vector[i+1]))

{NewBicluster(ColStart[i],ColEnd[i+1],RowNum[i],RowNum [i+1]);
 Vector. Push(NewBicluster);
}
```

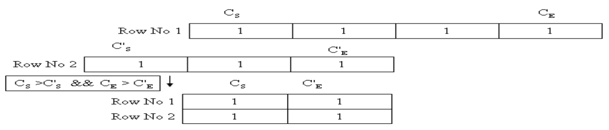


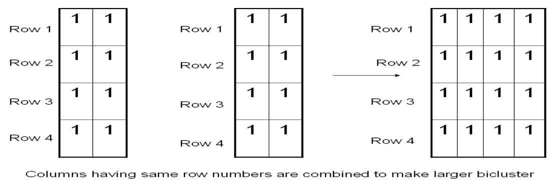
Fig. 15 CS >C'S && CE > C'E

C. Combining Column Biclusters:

The column biclusters whose columns are different but row numbers are same are combined on the basis of their row numbers. Figure 16 explains the scenario and pseudo code below explains it. The Bicluster entity for this case is little modified. It contains ColStart and ColEnd indices of both locations. RowNum vector indices are manipulated through index j.

Column Combining Pseudo code:

```
if(RowNum[j] of Vector[i] = = RowNum[j] of Vector[i+1])
{
 Bicluster (ColStart of Vector [i], ColEnd of Vector [i+1],ColStart[i+1],ColEnd[i+1], RowNum[j]);
  Vector.Push (Bicluster);
}
```


Columns having same row numbers are combined to make larger bicluster
Fig. 16 Columns having similar row numbers are combined to make one

VII. DIFFERENT SITUATIONS/EXAMPLES

We executed our proposed algorithm on different examples which we also checked through BiMax algorithm. Our approach produced the same biclusters as were produced by BiMax. The complexity of our approach is O(n*m) and reduces the extra computations of traversal of the matrix many times.

VIII. Implementation And Testing

We implemented the approach and tested it on various examples. Some of the generalized forms of tested examples are listed from figure 17 to figure 25.

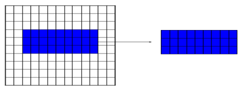


Fig. 17 Test case of Case 1

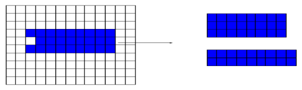


Fig. 18 Test case of Case 1, 6 and 7

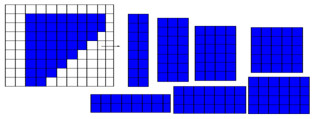


Fig. 19 Test case of case 1 & 5

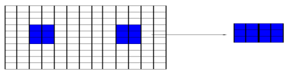


Fig. 20 Test case of case 1& column combining biclusters

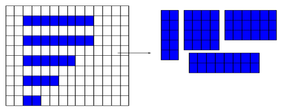


Fig. 21 2nd e.g. of test cases 1

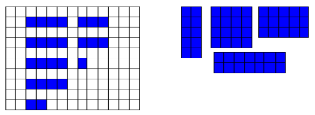


Fig. 22 Test case of case 1,5 & column combining biclusters

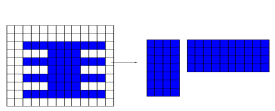


Fig. 23 Test case of case 2

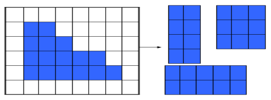


Fig.24 Test case of case 4

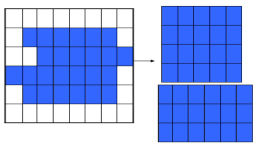


Fig.25 Test case of case 8,9 and1

Case 1,6 &7 are shown in fig 18. Case1& 5 are applied to get result of test case in fig 19& 21.Fig 20 of test cases for case 1 and column combining biclusters. Case 1, 5 and column combining are applied to get result in fig.22. Case 4 is used to build biclusters shown in fig.24 of the test cases. Case 8, 9 &1 built bicluster in test case shown in fig.25.

IX. Conclusions

This paper presented a brief overview of importance of biclustering techniques over the clustering techniques along with the discussion of some important biclustering methods. Problems in the well-know biclustering algorithm, BiMax, have been identified. A new algorithm, BiSim, with less complexity and a simpler approach has been presented. The algorithm through its test cases proved to be more efficient then the BiMax algorithm and avoids the extra recursive computations done by BiMax. Some of the important test cases have been presented to prove the working of the proposed algorithm.

Acknowledgment

Our thanks to the fall 2008 graduate class of advanced algorithms for critical and detailed discussions on BiMax algorithm and initial ideas towards the development of a new algorithm. We are grateful to faculty of engineering and applied sciences, Mohammad Ali Jinnah University, Islamabad for support in the research work.

References

[1] T.M.Murali and S. Kasif, *"Extracting Conserved Gene Expression Motifs from Gene Expression Data"*, *Pacific Symposium on Biocomputing*, vol. 8, pp. 77-88, 2003.
[2] S. Fodor , R. Rava, X. Huang , A. Pease, C. Holmes , and C. Adams, *Multiplexed Biochemical Assays with BiologicalChips*, *Nature*,vol. 364(6437),pp.555-6, 1993.
[3] M. Schen, D. Shalon, R.Davis and P.Brown, *Quantitative Monitoring of Gene Expression Patterns with a Complementary DNA Microarray, Science vol.* 270(5235), pp.467-70, 1995.
[4] S. C. Madeira and A. L. Oliveira, *Biclustering Algorithms for Biological Data Analysis: A Survey*, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* , 1, 24-45, 2004.
[5] A.Ben-Dor, B. Chor, R.Karp, and Z. Yakhini, "*Discovering Local Structure in Gene Expression Data: The Order-Preserving Sub-Matrix Problem"*, Proc. of the 6th Ann. Int'l Conf on Computational Biology, 1-58113-498-3, 49-57, 2002.
[6] M.B. Eisen, P.T.Spellman, P.O. Brown, and D. Botstein, *"Cluster Analysis and Display of Genome-Wide Expression Patterns",* Proc Nat1 Acad Sci USA, vol. 95(25): pp.14863-8, 1998.
[7] P. T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen,, P.O.Brown , D.Botstein, and Futcher, *Comprehensive Identification of ell Cycle-Regulated Genes of the Yeast Sacccharomyces Cerevisiae by Microarray Hybridization,* Molecular Biology of the Cell, 9~3273-3297, 1998.
[8] Y. Cheng and G. Church, *Biclustering of Expression Data*, *"Int'l Conf." on Intelligent Systems for Molecular Biology*, 93-103, 2000.
[9] T. Hastie, E. Tibshirani , and M.B Eisen, et al, *Gene Shaving as a Method for Identifying Distinct Sets of Genes with Similar Expression Patterns*, Genome Biol. ,Vol 1, p.RESEARCH0003.T, 2000
[10]G. Getz, E.Levine, E. Domany, (2000) *Coupled Two-way Clustering Analysis of Gene Microarray Data*, *PNAS*, vol. 97(22), pp. 12079–12804, 2000.
[11] S.Bergmann, J. Ihmels, N. Barkai ,*Iterative Signature Algorithm for the Analysis of Large-scale Gene Expression Data*, *Phys Rev E Stat Nonlin SoftMatter Phys*, 67(3), 031902, 2003.
[12] A .Tanay, R. Sharan and R. Shamir, *Discovering Statistically Significant Biclusters in Gene Expression Data*, *Bioinformatics*, 18, 136S-144, 2002.
[13]A. Prelic, S. Bleuler, P.Zimmermann, A. Wille, P. B¨uhlmann, W. Gruissem , L. Hennig, L. Thiele, and E. Zitzler , *A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data, Bioinfoinformatics* 22(9),1122-1129, 2006.
[14] J. A. Hartigan, "*Clustering Algorithms"*, *New York: John Willey and Sons, Inc,* 1975.
[15] J.H. Ihmels and S. Bergmann, *Challanges and Prospects in the Analysis of Large-Scale Gene Expression Data*, Briefings in Bioinformatics, vol. 5(4): pp.313-327, 2004
[16] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, The MIT Electrical Eng, and Computer Science Series, The MIT Press, second ed., 2001.
[17] C. Tang, L. Zhang, I. Zhang, and M. Ramanathan, "Interrelated Two-Way Clustering: An Unsupervised Approach for Gene Expression Data Analysis," Proc. Second IEEE Int'l Symp. Bioinformatics and Bioeng., pp. 41-48, 2001.
[18] S. Busygin, G. Jacobsen, and E. Kramer, "Double Conjugated Clustering Applied to Leukemia Microarray Data," Proc. Second SIAM Int'l Conf. Data Mining, Workshop Clustering High Dimensional Data, 2002.
[19] L. Lazzeroni and A. Owen, "Plaid Models for Gene Expression Data," technical report, Stanford Univ., 2000.