On Mining Micro-array data by Order-Preserving Submatrix

Lin Cheung   Kevin Y. Yip   David W. Cheung   Ben Kao   Michael K. Ng

Department of Computer Science,
The University of Hong Kong, Hong Kong.
{lcheung, ylyip, dcheung, kao, mng}@cs.hku.hk

Abstract

We study the problem of pattern-based subspace clustering. Unlike traditional clustering methods that focus on grouping objects with similar values on a set of dimensions, clustering by pattern similarity finds objects that exhibit a coherent pattern of rises and falls in subspaces. Applications of pattern-based subspace clustering include DNA micro-array data analysis, automatic recommendation systems and target marketing systems. Our goal is to devise pattern-based clustering methods that are capable of (1) discovering useful patterns of various shapes, and (2) discovering all significant patterns. We argue that previous solutions in pattern-based subspace clustering do not satisfy both requirements. Our approach is to extend the idea of Order-Preserving Submatrix (or OPSM). We devise a novel algorithm for mining OPSM, show that OPSM can be generalized to cover most existing pattern-based clustering models, and propose a number of extension to the original OPSM model.

Keywords*: Gene Expression, Data mining, Pattern-based clustering*

1. Introduction

The invention of DNA micro-array technologies has revolutionized the experimental study of gene expression. Thousands of genes are probed every day. Gene expression data analysis becomes one of the hottest topics in data mining, artificial intelligence, bioinformatics, and in the statistics community. Various data analysis techniques have been intensively studied.

Clustering has been one of the most popular methods of discovering useful biological insights from gene expression data. Many novel clustering techniques have been proposed. A problem that has attracted much interest lately is the discovery of clusters that are embedded in certain subspaces of high-dimensional data (such as gene expression data). The problem is known as subspace clustering. In this paper we explore the problem of pattern-based clustering — a special type of subspace clustering that uses *pattern similarity* as a measure of object distances.

In DNA micro-array data analysis, gene expression data is organized as matrices. In such matrices, a *row* carries the information of a *gene* and a *column* represents a *sample* for the experiment. The number in each cell records the *expression value* of a particular gene under a particular sample. In this paper we use the terms *object* and *dimension* (or *attribute*) to mean a row (gene) and a column (sample) of a dataset, respectively.

The objective of data clustering is to group together data points that are *close* or *similar* to each other in clusters. An important parameter to any clustering model is a distance (or similarity) measure. Typical distance functions include Euclidean distance, Manhattan distance, and cosine distance. For high-dimensional data, however, objects tend to exhibit strong similarity only over a (often unknown) subset of the attributes. Clustering data over the global set of attributes often fails to extract any meaningful clusters. This problem shows up in micro-array data analysis, which is typically high-dimensional.

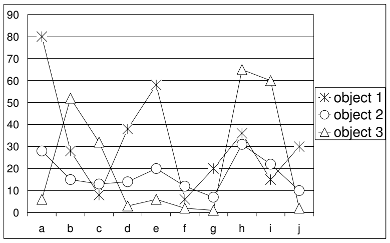

Figure 1. Raw data: 3 rows and 10 columns.

As an example, Figure 1 shows a set of 3 rows (lines) with 10 columns (labeled ‘a’ to ‘f’). The y-axis shows the expression values. If we consider the values from all 10 columns, there are no patterns observed. However, if we

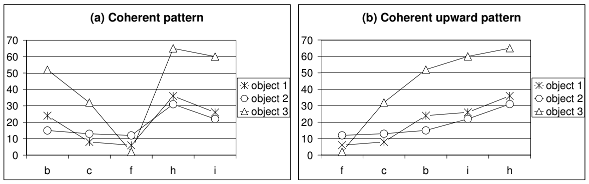

Figure 2. The 3 rows exhibits a coherent pattern in subspace.

select the set of columns b, c, f, h, i and show only the expression values for those columns (see Figure 2(a)), we observe something interesting: the expression values of the rows follow the same rise-and-fall pattern over the selected columns. Technically, we can consider the rows form a cluster in the *subspace b, c, f, h, i*. The example illustrates that traditional distance functions are sometimes inadequate in capturing correlations among rows.

Another way to observe the pattern that is shared by the rows is to rearrange the columns so that the expression values are listed in an ascending order. Figure 2(b) shows the values when the columns are rearranged according to the sequence *f, c, b, i, h*. We can see that the expression values are all increasing under the new column sequence. We call the sequence *f, c, b, i, h* an *order-preserving pattern*. The *length* of an order-preserving pattern is the number of columns in it; A row is a *supporting row* if its values exhibit an increasing order with respect to the column sequence; the *support* of a pattern refers to the number of supporting rows. In our example, the pattern *f, c, b, i, h* has a length of 5 and a support of 3. An order-preserving pattern together with the set of supporting rows form an *Order Preserving Sub-matrix*, or OPSM for short. Note that in our OPSM model, clusters may overlap. That is, rows could belong to multiple clusters. This model is reasonable for gene expression analysis and function prediction, since the same set of genes can have different functions under different samples.

In this paper our goal is to discuss the major challenges of the OPSM problem and to develop an efficient algorithm for solving it. In addition, we propose some computationally challenging variations of the OPSM problem. For those variations, we discuss potential solutions.

Subspace clustering is a computationally challenging problem. The complexity lies in the requirement of simultaneously determining both cluster members and relevant dimensions/attributes. Also, it is often difficult to determine the dimensionality of each cluster. The latter problem is particularly true for gene-expression analysis due to the lack of domain knowledge and the large number of attributes.

The OPSM problem is a very challenging subspace clustering problem. First, the number of potential order-preserving patterns grows exponentially with respect to the number of attributes. A dataset D with n attributes has $\sum_{i=2}^{n} n!/(n-i)!$ potential order-preserving patterns. For DNA micro-array datasets, there could be tens or even hundreds of attributes. Examining each OPSM as a potential cluster is clearly infeasible. Effective pruning techniques are needed.

Another challenge to the OPSM problem is that the number of potential OPSM is huge. Recall that an OPSM consists of a set of rows and a set of columns (arranged in a certain sequence). We note that any subset of those rows plus any subset of those columns form a valid OPSM. These derived OPSMs, or subclusters, however, are redundant. In this paper we focus on mining *maximal* OPSMs, that is, those that are not proper subclusters of others.

The rest of the paper is structured as follows. In Section 2, we review some related work. Section 3 gives a formal definition of the OPSM model. Section 4 discusses our OPSM algorithm in details. In Section 5, we propose some advanced pruning methods for solving the OPSM problem. Section 6 discusses some interesting variations of the OPSM problem. A generalization of the pattern-based clustering problem is discussed in section 7. Section 8 concludes the paper.

2. Related works

In [2], Cheng and Church suggest modeling DNA microarray dataset as a matrix, where each gene is represented as a row and each sample is represented as a column. They introduced the bicluster concept as a measure of the coherence of the genes and samples. A cluster is defined as a submatrix (a subset of the rows and a subset of the columns that are not necessarily contiguous). Let O be the set of row and C the set of columns. Let $I \subset O$ and $J \subset C$ be a subset of rows and a subset of column, respectively. The pair (I, J) specifies a submatrix $A_{I \times J}$ with a mean squared residue score defined by $H(I, J) = 1/(|I||J|) \sum_{i \in I, j \in J} (d_{ij} - d_{iJ} - d_{Ij} + d_{IJ})^2$. The term $d_{iJ} = 1/|J| \sum_{j \in J} d_{ij}$ represents the row mean, $d_{Ij} = 1/|I| \sum_{i \in I} d_{ij}$ represents the column mean, and $d_{IJ} = 1/|I||J| \sum_{i \in I, j \in J} d_{ij}$ is the mean over the whole submatrix A_{IJ}. A submatrix A_{IJ} is called a δ-bicluster if $H(I, J) \leq \delta$ for some $\delta > 0$. A greedy algorithm is proposed to discover the cluster with the lowest score. Yang et al. [12] proposed another algorithm that tries to find multiple clusters at the same time. The Plaid model [7] and the Spectral model [6] were proposed as improvement over the bi-cluster model.

A different model pCluster was proposed by Wang et al. [11]. A non-contiguous submatrix is a cluster if for any pair of rows i_1 and i_2, and any pair of columns j_1 and j_2 in the cluster, $|(d_{i_1,j_1} - d_{i_1,j_2}) - (d_{i_2,j_1} - d_{i_2,j_2})| \leq \delta$, where

$d_{x,y}$ represents the value in row x and column y. In other words, the value change across two attributes must not vary a lot in different rows. An algorithm for finding pClusters is proposed in the paper. Two improved algorithms MaPle [4] and SeqClus [10] have been proposed afterwards.

While many different models have been proposed, most of them are quite restrictive. In order to identify more general clusters, the OPSM model has been proposed by Ben-Dor et al. [3]. A non-contiguous submatrix is an OPSM cluster if there exists a permutation of the columns such that in the resulting permuted matrix, the values in each row are monotonically non-decreasing. This means for any two rows i_1 and i_2 and any two columns j_1 and j_2 in the original submatrix, the signs of $d_{i_1,j_1} - d_{i_1,j_2}$ and $d_{i_2,j_1} - d_{i_2,j_2}$ are the same. In other words, the model only requires the rows to have the same direction of response across different columns, but the absolute magnitudes of response are unimportant. In [3], a greedy algorithm was proposed to identify some number of good OPSMs from a dataset. The algorithm, however, does not guarantee that all OPSMs are found, nor the best ones are found.

An extension of OPSM, namely OP-Cluster [8, 9] is proposed by J. Liu et al. Given a user-specified error threshold δ. Columns with their values differ within δ are grouped into an equivalent class. The order of columns within an equivalent class is ignored. They proposed a tree structure for storing all existing patterns and a depth-first search algorithm for mining all error-tolerated clusters. However, the time and space complexities of the algorithm increase exponentially with the number of dimensions.

Our goal is to develop an algorithm that can efficiently identify *all* OPSMs in a dataset. We also propose a number of variations to the cluster definition, which are of practical values.

3. Problem Definition

In this section we give a formal description of the OPSM problem. Consider a gene-expression dataset D, represented as a matrix. We use O and C to denote the set of rows and columns in D, respectively. We use $d_{i,j}$ to denote the entry of D in row i and column j.

A *cluster* S is a submatrix of D formed by a subset of $n_S (\geq 2)$ rows and a subset of $m_S (\geq 2)$ columns of D. Rows and columns in S need not be contiguous in D. The rows in S are referenced by their row indices in D, each of which is a distinct integer in $\{1, 2, ..., n_D\}$. The set of row indices of S is denoted as R_S. Columns in S are similarly referenced. The set of columns in S is denoted by C_S. We use $s_{i,j}$ to denote an entry in S with i, j being the references w.r.t. the dataset D. For example, $s_{2,3}$ refers to the entry in S that is taken from the 2nd row and the 3rd column of D. The term *row index* refers to the location of a row in D rather than in the cluster concerned (such as S). The same holds for the term *column index*.

The columns in S are enclosed in curly brackets, e.g., $C_S = \{c_1, c_2, ..., c_{m_S}\}$. A *sequence* $\hat{C_S}$ of the columns in S is enclosed in angled brackets, e.g., $\hat{C_S} =< c_1, c_2, ..., c_{m_S} >$. The columns in a sequence is totally ordered. For the basic OPSM problem, a cluster is a set of rows and a set of columns such that entries in every row are increasing w.r.t. a particular column sequence. Hence, the order the columns is important. A cluster S is thus written as $S = (R_S, \hat{C_S})$.

A *permutation* of a sequence is a reordering of the columns inside the sequence. For example, $< x, y, z >$ is a permutation of $< z, x, y >$. If the columns of a cluster S is permuted to form another cluster P, both clusters have exactly the same columns, but the order of their columns can be different. In other words, $C_S = C_P$ but $\hat{C_S}$ may not equal $\hat{C_P}$.

Definition 1 *A cluster S is an* OPSM *if there exists a permutation of the columns such that in the permuted cluster P, $p_{i,j} \leq p_{i,j+1}$ for all $i \in \{1, 2, ..., n_P\}$ and all $j \in \{1, 2, ..., m_P - 1\}$. If a cluster satisfies the requirement without the need to permute its columns, it is called an* in-sequence OPSM.

A cluster S is a *subcluster* of a cluster S' if $R_S \subseteq R_{S'}$ and $C_S \subseteq C_{S'}$. A cluster S is a *proper subcluster* of a cluster S' if S is a subcluster of S' and either $R_S \neq R_{S'}$ or $C_S \neq C_{S'}$.

Definition 2 *An OPSM is a* maximal OPSM *if it is not a proper subcluster of any OPSM.*

An OPSM $S = (R_S, C_S)$ is a *row-maximal OPSM* if there does not exist a cluster $S' = (R_{S'}, C_S)$ such that $R_S \subset R_{S'}$. *Column-maximal OPSM* is defined similarly.

Given a data matrix D, the basic OPSM problem is to find all in-sequence OPSMs in D.

Note that for any pair of column indices j_1 and j_2, there are at most two maximal OPSMs of the form $S = (R_S, < j_1, j_2 >)$. One of them contains all rows i where $d_{i,j_1} \leq d_{i,j_2}$ and the other contains all rows i' where $d_{i',j_1} \geq d_{i',j_2}$. The larger cluster among the two has at least $\lceil \frac{|O|}{2} \rceil$ rows. As a result, for a dataset D with n_D rows, every length-2 column sequence must have $n_D/2$ supporting rows on average. This argument shows that short patterns are too numerous and uninteresting. Also, some patterns (column sequences) may have only a few of supporting rows. These patterns are not very interesting either since they are not statistically significant. We thus modifiy the basic OPSM problem so that only those OPSMs with a significant pattern length and a significant support are reported.

Problem Statement 1 *(Maximal size-constrained OPSM problem): Given a data matrix D, a supporting row thresh-*

old n_{min}, and a column threshold m_{min}, find all maximal in-sequence OPSMs S in D such that $n_S \geq n_{min}$ and $m_S \geq m_{min}$.

4. Algorithm for finding OPSMs

In this section, we propose a new algorithm for exploring all maximal size-constrianed in-sequence OPSMs.

4.1. Algorithm

Our algorithm is similar to the Apriori algorithm for mining Association rules [1]. In Apriori, it mines a transaction database D' and discovers all *frequent* itemsets. First, it generates all itemsets with 2 items, we called them size-2 itemsets. Then it scans the D' and counts transactions contain the itemset, it is referred as *support*. An itemset is frequent if it's support greater or equal to a user specified threshold. For all integer k > 2, it generates size-k itemsets by concatenating two size-(k-1) itemsets with (k-2) items in common. It scans D' for counting the support for each itemset. It terminates when there are no frequent size-(k+1) itemsets can be generated.

Property 1 *(A priori property): A cluster S is an OPSM if and only if all proper subclusters of S are also OPSM.*

Proof ($\Rightarrow$:) Suppose a cluster $S = (R_S, \hat{C_S})$ is an OPSM and a cluster $P = (R_S, \hat{C_P})$ is a corresponding in-sequence OPSM formed by permuting the columns of S. By definition, $p_{i,j} \leq p_{i,j+1}$, for all $i \in \{1, 2, ..., n_S\}$ and all $j \in \{1, 2, ..., m_S - 1\}$. The inequalities remain valid if some rows and columns are deleted from P. Therefore for any cluster $S' = (R_{S'}, C_{S'})$ where $R_{S'} \subseteq R_S$ and $C_{S'} \subseteq C_S$, an in-sequence OPSM P' can be formed by removing from R_S all row indices that are not in $R_{S'}$, and from $C_{S'}$ all column indices that are not in $C_{S'}$. Since S' can be formed by permuting the rows and columns of P', S' is an OPSM.

($\Leftarrow$:) Suppose a cluster S and all its proper subclusters are OPSMs. Consider a cluster S' is formed by removing the first row of S. Since S' is an OPSM, there exist some integer $j \in X$ such that $s_{i,j}$ is the smallest element in row i of $S, \forall i \in \{2, 3, ..., n_S\}$. For the sake of contradiction, suppose the smallest element in the removed row exists at column $j' \notin X$. This means $s_{1,j'} < s_{1,j}, \forall j \in X$. Since the cluster formed by keeping only the first and the i-th rows of S is an OPSM for all $i \in \{2, 3, ..., n_S\}$, $s_{i,j'} \leq s_{i,j}, \forall j \in X$. But by the definition of X, $s_{i,j} \leq s_{i,j'}$. This implies $s_{i,j'} = s_{i,j}$, which is a contradiction since $j' \notin X$. Therefore, there must exist an integer $j'' \in X$ s.t. $s_{1,j''}$ is the smallest element in the first row, and thus all rows of S. Consider a cluster T formed by removing the j''-th column of S. T is an OPSM since it is a proper subcluster of S. By adding the removed column back to T, the resulting cluster S'' must also be an OPSM since all rows of it have the smallest element at the added column. Since S can be formed by permuting the columns of S'', S'' is an OPSM.

Our algorithm shares similar heuristic with Apriori algorithm, but with an additional constraint, the columns(the same role of items in Apriori) selected have an order.

Property 2 *(Transitivity): If $S_1 = (R_{S_1}, < x_1, x_2, ..., x_i, y_1, y_2, ..., y_j >)$ and $S_2 = (R_{S_2}, < y_1, y_2, ..., y_j, z_1, z_2, ..., z_k >)$ are two row-maximal in-sequence OPSMs and $R_{S_1} \bigcap R_{S_2}$ contains n_{min} or more indices, then $S = (R_{S_1} \bigcap R_{S_2}, < x_1, x_2, ..., x_i, y_1, y_2, ..., y_j, z_1, z_2, ..., z_k >)$ is a row-maximal in-sequence OPSM.*

Proof: let S' be the row-maximal in-sequence OPSM with $\hat{C_{S'}} = < x_1, x_2, ..., x_i, y_1, y_2, ..., y_j, z_1, z_2, ..., z_k >$. If a row index is in R_S, i.e., in both R_{S_1} and R_{S_2}, it must also be in $R_{S'}$. Therefore S is an in-sequence OPSM. If a row index i is not in R_S, then it is either not in R_{S_1} or R_{S_2}. In the former case, i must not be in $R_{S'}$ because if it is in $R_{S'}$, then $(R_{S_1} \wedge i, C_{S_1})$ would be an OPSM as it is a proper subcluster of S', which is a contradiction since it implies that S_1 is not row-maximal. The same argument holds for row indices that are not in R_{S_2}. Therefore, S is row-maximal.

According to the transitivity property, we can form S from S_1 and S_2 without the need rescan the whole data matrix to check for the row indices that are in R_S.

Our algorithm takes a data set D with size $|O| \times |C|$, a column threshold m_{min}, and a supporting rows threshold n_{min} as input. It finds all row-maximal in-sequence OPSMs with two columns. For any pair of clusters S_1 and S_2, where the last column index in $\hat{C_{S_1}}$ equals the first column index in $\hat{C_{S_2}}$. We create a new cluster S with $R_S = R_{S_1} \bigcap R_{S_2}$ and $\hat{C_S}$ equals $\hat{C_{S_1}}$ with the last column index in $\hat{C_{S_2}}$ appended to the end. We only keep S if $|R_S| \geq 2$. After creating all row-maximal in-sequence OPSMs with three columns, repeat the same procedures to form row-maximal in-sequence OPSMs with four columns, and so on, until no more clusters can be formed. Whenever all clusters with $k+1$ columns are created, the algorithm performs a maximality test on each cluster S with k columns against all clusters with $k+1$ columns. S is maximal if there exists no cluster S' with $k+1$ columns such that $R_S = R_{s'}$ and $C_S \subset C_{S'}$. All such S' are added to the result set. Others are discarded.

Completeness of the algorithm: For each row-maximal in-sequence OPSM $S = (R_S, < c_1, c_2, ..., c_{n_S} >)$, there must exist a row-maximal in-sequence OPSM $S'_i = (R_{S'_i}, < c_i, c_{i+1} >)$, where $R_S \subseteq R_{S'_i}$, for all $i \in \{1, 2, ..., n_S - 1\}$, due to the apriori property. Therefore according to the algorithm, the clusters $S''_i = (R_{S''_i}, < c_i, c_{i+1}, c_{i+2} >)$ must be created for all $i \in \{1, 2, ..., n_S - 2\}$, where $R_S \subseteq R_{S''_i}$. Iteratively, a cluster $S^* = (R_{S^*}, < c_1, c_2, ..., c_{n_S} >)$ must be created where $R_S \subseteq R_{S^*}$. Since S is row-maximal, R_{S^*} must equal R_S, so S must be identified by the algorithm.

Correctness of the algorithm: By the transitivity property, all clusters being formed are row-maximal in-sequence OPSMs. The ones being added to the result set are further proved to be column-maximal due to the maximality test and the fact that all row-maximal in-sequence OPSMs with one more column are discovered.

4.2. Data Structure

In this sub-section, we propose a novel data structure that are capable efficient processing of (1) identifying all pairs of length-k column sequences where the last $k-1$ indices of the first sequence equal the first $k-1$ indices of the second sequence, and (2) intersecting two sets of row indices. It is referred as *Head-Tail Trees*.

We build a head tree and a tail tree in each iteration. Both of them are balanced tree. Head tree and tail tree store all clusters according to the first $k-1$ column indices and the last $k-1$ column indices respectively. Each tree node contains a column index as key. Child nodes are ordered according to the column indices lexicographically. Each set of $k-1$ column indices is represented by a path from the root to a leaf. It is referred by *path sequence*. Two sets share the first x nodes if their first x indices are the same. Each leaf node in the head tree contains pointers to clusters whose first $k-1$ column indices equals to the path sequence. Pointers to clusters are stored in tail tree leaf node similarly. Each cluster stores its row indices by a list of row indices.

Suppose we get a set of maximal in-sequence OPSMs of sequence length n_S. For each cluster $S = (R_S, < c_1, c_2, ..., c_{n_S} >)$, we insert the cluster along the path $< c_1, c_2, ..., c_{n_S - 1} >$ into the head tree and the path $< c_2, c_3, ..., c_{n_S} >$ into the tail tree. Add a pointer to S from both leaf nodes.

By traversing Head-Tail trees in pre-order, new clusters are generated at leaf nodes. If the current leaf node at head tree has path sequence lexicographically smaller(larger) than that of the current node of the tail tree, we continue the tree traversal of the head(tail) tree. If two path sequences are equal, we try to join each pair of clusters in the two lists linked by the two nodes to form new clusters.

Suppose S_1 is a cluster in the list linked by a head tree node, and S_2 is a cluster in the list linked by a tail tree node, where $S_1 = (R_{S_1}, < c_2, ..., c_{k+1} >)$ and $S_2 = (R_{S_2}, < c_1, c_2, ..., c_k >)$ respectively. We skip this pair if $c_1 = c_{k+1}$. Otherwise, we intersect the 2 lists of row indices. If the resulting list has n_{min} or more row indices, insert $< c_1, c_2, ..., c_{k+1} >$ into the head and tail trees for the $k+1$ iteration, and add the pointer to the new cluster from both leaf nodes.

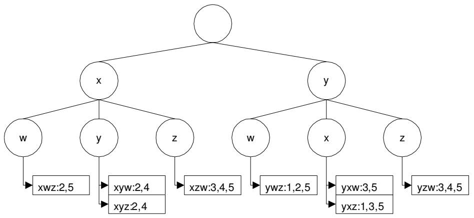

Figure 3. The head tree for clusters with 3 columns.

We can optimize the set intersection operations by storing the row indices in bit bit vectors of length $|C|$. The merge joins are replaced by fast bitwise AND operations.

We identify that the head tree built in the 1st iteration is sufficient for forming new clusters. We can only check if the head tree leaf node path sequence equals to the tail tree 1st index of path sequence. We save insertion time by omitting the insertions of new path sequences in the head tree. Since the head tree is fixed, we can even store it in a simple list. However a larger number of unnecessary join operations may be performed. Suppose a cluster with column sequence $< abc >$. If both trees are updated, a new cluster with column sequence $< abcd >$ will be generated only if there exists a cluster with column sequence $< bcd >$. If we only update the tail tree, the join operations consider a bigger set of clusters with path sequence $< cd >$.

4.3. Example

	w	x	y	z
1	7	8	6	9
2	3	0	2	6
3	9	3	2	8
4	3	0	1	2
5	5	4	2	5

$\hat{C}_S$	R_S
wz	1,2,5
xw	2,3,4,5
xy	2,4
xz	1,2,3,4,5
yw	1,2,3,4,5
yx	1,3,5
yz	1,2,3,4,5
zw	3,4,5

$\hat{C}_S$	R_S
xwz	2,5
xyw	2,4
xyz	2,4
xzw	3,4,5
ywz	1,2,5
yxw	3,5
yxz	1,3,5
yzw	3,4,5

$\hat{C}_S$	R_S
yxzw	3,5

Table 1. Data Set D with Row-maximal in-sequence OPSMs.

The resulting trees are illustrated in Figure 3 to Figure 4. There are some special cases to note. First, there is no clusters with $\hat{C}_S = wx$ since only row 1 supports it. Second, there is no clusters with $\hat{C}_S = wy$ since no rows support it. Third,the value 5 appears twice in row 5, so it appears in both clusters with $\hat{C}_S = wz$ and $\hat{C}_S = zw$.

5. Pruning and Optimizations

In this section, we propose several efficient rules to prune unpromising pattern using special properties of OPSM.

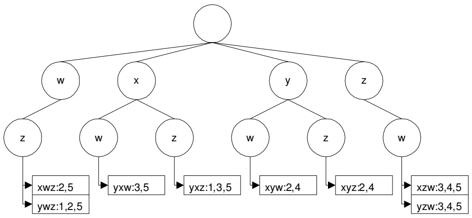

Figure 4. The tail tree for clusters with 3 columns.

Optimization 1: For each pattern, we count the rows with column sequence starting with the first column index of the pattern, it is referred as *first count*. Similarly, *last count* is kept for last column index of the pattern.

Suppose a dataset has 4 columns, namely a, b, c, d, and there is a row o follows a pattern $< badc >$. It contributes 1 to first count of each pattern start with b, and last count of each pattern end with c. Suppose a pattern $< ac >$ has support of 10 and a last count of 4, then the supporting rows of pattern $< acx >$ (where x = b or d) is at most 6. In other words, if n_{min} is 7 or more, we can not have a cluster with pattern $< acx >$. Similarly, suppose a pattern $< cd >$ has first count of 5, we cannot have a pattern $< xcd >$ (where x = a or b) if threshold is 6 or more.

This optimization technique assumes there has no duplicate values in each row. A pre-processing step is needed to eliminate all duplicate values in each row. It can be done by appending the column index to the end of each value.

Given pattern p_r has support s_{p_r}, first count f_{p_r} and last count l_{p_r}; pattern p_s has support s_{p_s}, first count f_{p_s} and last count l_{p_s}. A pattern p_{new} is generated by extending p_r with p_s, with unknown support $s_{p_{new}}$, first count $f_{p_{new}}$, and last count $l_{p_{new}}$. According to the heuristic above, we know $s_{p_{new}}$ is at most min(s_{p_r} - l_{p_r}, s_{p_s} - f_{p_s}), $f_{p_{new}}$ is at most f_{p_r} and at least $s_{p_{new}}$ - (s_{p_r} - f_{p_r}), and $l_{p_{new}}$ is at most l_{p_s} and at least $s_{p_{new}}$ - (s_{p_s} - l_{p_s}). With the help of these formulas, we can skip support counting of patterns with maximum support less than user inputted threshold. Moreover, we may use the upper bounds of $f_{p_{new}}$ and $l_{p_{new}}$ to perform pruning in the next iteration.

Optimization 2:For example, the supporting rows of $< ab >$ and the supporting rows of $< ba >$ must sum up to $|O|$. Similarly, if we have created node $< ab >$ and know it's support, we know it must equal the sum of supports of $< abc >$, $< acb >$ and $< cab >$. If we have the supports of $< ab >$, $< abc >$ and $< acb >$, we can calculate the support of $< cab >$ eventually. In another view, support of $< cab >$ can also be inferred from support counts of $< ca >$, $< cba >$ and $< bca >$. In all these cases, we can infer the support of some patterns by those of others. We only need one of these combinations to get the support of $< cab >$. It is good to have multiple ways because some supports may not be available. For example, if $< abc >$ is infrequent, we do not have its support. Then, we cannot use the combination of $< ab >$, $< abc >$ and $< acb >$ for the inference. However, a longer pattern requires supports of more patterns to perform a single pruning.

6. Variations of OPSM

In this section, we propose several possible biologically significant variations of OPSM. They are Sign-constrained problem, Sign-constrained Bidirectional problem, and Error-tolerated problem. To ease understanding, we provide examples to illustrate clustering results using the following data matrix D' for different variation problems. We also propose simple methods for solving the problems.

	a	b	c	d
o_1	10.3	21.8	29.8	2.3
o_2	3.8	42.1	59.2	3.5
o_3	10.4	13.8	15.9	-2.5
o_4	51.1	58.2	59.9	68.1
o_5	53.9	41.8	39.8	18.3
o_6	0.3	21.8	20.9	2.3

Table 2. Data matrix D'.

6.1. Sign-constrained problem

In DNA micro-array, each entry $d_{i,j}$, representing the expression level of gene i in sample j, is derived by comparing the expression level of gene i, the gene of interest, and expression level of a reference gene. A positive value means the gene i is over-expressed at sample j. A negative value implies the gene i is under-expressed at sample j. Their biological meanings are totally different.

Suppose a gene g_a and a gene g_b exhibit the same rise and fall patterns on a set of samples C'. g_a is over-expressed on C' and g_b is under-expressed on a subset of C'. We can not group g_a and g_b into the same cluster. Motivated by this, We come up with the Sign-constrained problem.

Variation Problem 1 *(Sign-constrained problem): Given a data matrix D, find all in-sequence OPSMs S in D such that for all $j \in \{1, 2, ..., m_S\}$, $sign(s_{1,j}) = sign(s_{2,j}) = ... = sign(s_{n_D,j})$, where $sign()$ is the sign function defined as follows:*

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (1)$$

Example: Considering only rows $\{o_1, o_2, o_3, o_4\}$ data matrix D', there exists a sign-constrained cluster $S_1 = (\{o_1, o_2, o_3, o_4\}, < abc >)$. Note that,

$S_1 = (\{o_1, o_2, o_3\}, < dabc >)$ is a maximal size-constrained cluster, but not a sign-constrained cluster.

The sign consistent constraint verification can be optimized by using a bitmap. Create a bitmap B of size n × m, where $B_{i,j}$ stores the sign of expression level of gene i in sample j. The verification can be replaced by fast bitwise NOT XOR operations.

6.2. Sign-constrained Bidirectional problem

Given a sign-constrained maximal OPSM, we know all genes within this cluster with rows(genes) O_c exhibit a sequence C_s. It is biological interesting to know if there exists any genes O_d have the reverse sequence of C_s, as the activities of genes O_d suppresses the activity of genes O_c, and vice versa. According to this argument, two genes have reverse expression sequences are highly related.

Variation Problem 2 *(Sign-constrained Bidirectional problem): Given a data matrix D, find all in-sequence sign-constrained bi-directional OPSMs in D.*

A cluster S is a sign-constrained bi-directional OPSM if (1) it is a sign-constrained maximal OPSM, and (2) it can be divided into of 2 subclusters P_{up} and P_{down} where P_{up} contains all rows $i \in \{1, 2, ..., n_{P_{up}}\}$, $p_{i,j} \leq p_{i,j+1}$ for all $j \in \{1, 2, ..., m_{P_{up}} - 1\}$. P_{down} contains all rows $i \in \{1, 2, ..., n_{P_{down}}\}$, $p_{i,j} \geq p_{i,j+1}$ for all $j \in \{1, 2, ..., m_{P_{down}} - 1\}$. $n_{P_{up}}$ and $n_{P_{down}}$ sum up to n_P.

Example: Considering D', $n_{min} = 1$, and $m_{min} = 3$, one of the sign-constrained bi-directional OPSMs is $S_1 = (\{o_1, o_2, o_3, o_4, o_5\}, < abc >)$. (Note: $(\{o_1, o_2, o_3, o_4\}, < abc >)$ is a sign-constrained maximal OPSM.)

Given we discovered all sign-constrained maximal OPSMs in dataset D by the Head-tail tree approach, we can reuse the tail tree to see if we can find any genes exhibit a reverse sequence.

Example: Suppose a pattern P is a sign-constrained maximal OPSM with pattern $< abcd >$. We can traverse the tail tree by the path $< cba >$ to check if there exist any row follows expression pattern $< dcba >$.

6.3. Error-tolerated problem

In [5], authors claimed that the measurements of expression values in DNA microarrays may have errors. It initiates our proposal of a robust error-tolerated OPSM model.

Variation Problem 3 *(Error-tolerated problem): Given a data matrix D and an error threshold ϵ, find all in-sequence ϵ-tolerated OPSMs in D.*

A cluster S is a ϵ-tolerated OPSM if there exists a permutation of the columns such that in the permuted cluster P, $p_{i,j+1} - p_{i,j} \geq -\epsilon$, for all $i \in \{1, 2, ..., n_P\}$ and all $j \in \{1, 2, ..., m_P - 1\}$.

Example: Considering D' and ϵ be 2, there exists an error-tolerated clusters $S_1 = (o_1, o_2, o_3, o_4, o_5, o_6, < abc >)$. S_1 cannot be discovered in orginial OPSM.

We propose a post-processing solution for the Error-tolerated problem. We first generate all size-constrained maximal OPSMs. The input space is smaller here, as it only includes maximal OPSMs fullfilled the threshold requirements. An error-tolerated cluster c_{new} with pattern of length k+1 is generated only if there exists 2 clusters with pattern of length k, and these two patterns(sequence) differ at one position only. Moreover, c_{new} should have at least n_{min} rows support (see Table 4).

```
Let $S$ be the set of all maximal OPSMs in data $D$.
Let $maxLength$ be the length of the longest pattern exists in $S$.
For k = $m_{min}$ to maxLength do
    For each $c_i \in S$ with pattern of length k
        Let $p_{c_i}$ be the pattern of $c_i$, $p_{c_i} = < p_{i_1}, p_{i_2}, \ldots, p_{i_k} >$
        Find the set of maximal OPSMs $S_{k_i}$, where each cluster $c_j$ belongs to $S_{k_i}$, $c_j$'s pattern $p_{c_j} = < p_{j_1}, p_{j_2}, \ldots, p_{j_k} >$, $p_{c_i}$ and $p_{c_j}$ are the same in k-1 positions, and position g is the only position 2 patterns differ
        $p_{i_t} = p_{j_t} \forall t \in 1, 2, \ldots, g-1, g+1, \ldots k$
        for each $c_j \in S_{k_i}$
            $R = R_{c_i} \cap R_{c_j}$;
            $p_1 = < p_{i_1}, p_{i_2}, \ldots, p_{i_g}, p_{j_g}, p_{i_k} >$
            $p_2 = < p_{i_1}, p_{i_2}, \ldots, p_{j_g}, p_{i_g}, p_{i_k} >$
            $R_1$ and $R_2$ be 2 empty set of rows.
            for each row $r_v \in R$
                if $(d_{v,p_{j_g}} - d_{v,p_{i_g}} \geq -\epsilon)$
                    $R_1 = R_1 \cup r_v$;
                if $(d_{v,p_{i_g}} - d_{v,p_{j_g}} \geq -\epsilon)$
                    $R_2 = R_2 \cup r_v$;
            if $(|R_1| \geq n_{min})$
                $c_{new_1} = R_1, p_1$ or
                insert $c_{new_1}$ into $S$;
            if $(|R_2| \geq n_{min})$
                $c_{new_2} = R_2, p_2$
                insert $c_{new_2}$ into $S$;
```

Table 3. Algorithm for finding Error-tolerated clusters.

7. Generalization

As we mentioned earlier, there are many diverse models for pattern-based clustering. In different applications and for different users, different types of patterns may be needed. The measurement of quality of patterns and clusters also differs in different models. For example, both the pClustering model and the OPSM model are quite different in technical details, but they share similar philosophy. However, such a one method per variation approach may not be effective. In this section, we generalize OPSM model and propose a new generic clustering model which includes most previously proposed pattern-based clustering models(e.g. Biclustering model, pCluster model, OPSM model). Moreover, we prove some properties in the generic model.

Definition 3 *A matrix $S = (R_S, \hat{C_S})$ is a* Pattern-based Cluster *if and only if it satisfies the following inequalities:*

$$\forall i_1, i_2 \in \{1, 2, ..., n_S\}, \forall j_1, j_2 \in \{1, 2, ..., m_S\}, s.t. j_1 < j_2,$$

$$\alpha \leq (f(S_{i_1,j_2}) - f(S_{i_1,j_1})) - (g(S_{i_2,j_2}) - g(S_{i_2,j_1})) \leq \beta$$

By setting specific constraints, the generic model can fit previously proposed specific definitions (refer Table 4).

	f(x)	g(x)	α	β
Perfect δ-cluster	x	x	0	0
pCluster, shifting	x	x	-δ	δ
pCluster, scaling	log x	log x	-δ	δ
OPSM	x	0	0	$+\infty$

Table 4. Constriants for Specific Clustering Model.

7.1. Anti-monotonicity

Suppose a matrix has some rows and columns do not satisfy the inequality, then they remain to violate the inequality when new rows or columns are added to the matrix. This means all the clusters defined above intrinsically have the anti-monotonic property: a matrix cannot be a cluster if any of its proper submatrixes is not a cluster. Notice that here we involve both rows and columns in the definition of the anti-monotonic property. We can even observe a stronger property: a matrix is a cluster (according to any of the above definitions) if and only if all its proper submatrices are clusters. The reason is quite trivial: an inequality that holds in the matrix S must also hold in some of its submatrices, while an inequality that holds in a submatrix must also hold in matrix S.

An imperfect δ-cluster is not anti-monotonic because it does not enforce the constraints locally for any pairs of rows and columns, but only requires the whole cluster globally produces an aggregate score lower than a certain threshold.

7.2. Transitivity

As defined above, the transitivity property holds if for any two clusters S_1 = (R_S,$<$ x_1, x_2, ..., x_i, y_1, y_2, ..., y_j $>$) and S_2 = (R_S,$<$ y_1, y_2, ..., y_j, z_1, z_2, ..., z_k $>$), S_3 = (R_S,$<$ x_1, x_2, ..., x_i, y_1, y_2, ..., y_j, z_1, z_2, ..., z_k $>$) must also be a cluster.

We can start with the following simpler form: for any two clusters $S_1 = (R_S, < x, y >)$ and $S_2 = (R_S, < y, z >)$, $S_3 = (R_S, < x, y, z >)$ must also be a cluster. Now,

$$\alpha \leq (f(S_{i_1,j_2}) - f(S_{i_1,j_1})) - (g(S_{i_2,j_2}) - g(S_{i_2,j_1})) \leq \beta \quad (2)$$

$$\alpha \leq (f(S_{i_1,j_3}) - f(S_{i_1,j_2})) - (g(S_{i_2,j_3}) - g(S_{i_2,j_2})) \leq \beta \quad (3)$$

$$(2) + (3):$$

$$2\alpha \leq (f(S_{i_1,j_3}) - f(S_{i_1,j_1})) - (g(S_{i_2,j_3}) - g(S_{i_2,j_1})) \leq 2\beta \quad (4)$$

This means the simpler form holds if both α and β are either 0 or unbounded ($\pm\infty$). Perfect δ-cluster, and our OPSM cluster definitions all fall into this category. The complex form can then be proved by arbitrarily picking y_1 as j_2, and try each $x_{i'}(1 \leq i' \leq i)$ as j_1 and each $z_{k'}(1 \leq k' \leq k)$ as j_3.

8. Conclusions

In many applications including DNA array analysis, rows manifest consistent patterns on a subset of columns even they are not close in terms of distance. In this paper, we analyzed the OPSM model, which aimed at capturing the consistent tendency by a subset of rows in a subset of dimensions in high dimensional space. We proposed a Head-Tail Trees structure and an Apriori-like algorithm that can discover all OPSMs. We also discussed some advanced pruning methods on the OPSM model, some variations of the OPSM problem, and a generalization model of pattern-based clustering.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *VLDB*, 1994.

[2] Y. Cheng and G. M. Church. Biclustering of expression data. *ISMB*, 2000.

[3] A. B.-D. et. al. Discovering local structure in gene expression data: the order-preserving submatrix problem. *RECOMB*, 2002.

[4] J. P. et. al. MaPle: A fast algorithm for maximal pattern-based clustering. *ICDM*, 2003.

[5] J. P. B. et. al. Significance and statistical errors in the analysis of dna microarray data. *PNAS*, 2002.

[6] Y. K. et. al. Spectral biclustering of microarray cancer data: Co-clustering genes and conditions. *Genome Research*, 13(4):703–716, 2003.

[7] L. Lazzeroni and A. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12:61–86, 2002.

[8] J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. *ICDM*, 2003.

[9] J. Liu, J. Yang, and W. Wang. Biclustering in gene expression data by tendency. *CSB*, 2004.

[10] H. Wang, F. Chu, W. Fan, P. Yu, and J. Pei. A fast algorithm for subspace clustering by pattern similarity. *SSDBM*, 2004.

[11] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. *SIGMOD*, 2002.

[12] J. Yang, H. Wang, W. Wang, and P. Yu. Enhanced biclustering on expression data. *BIBE*, 2003.