Enhanced pClustering and Its Applications to Gene Expression Data

Sungroh Yoon
Stanford University
Stanford, USA
sryoon@stanford.edu

Christine Nardini
University of Bologna
Bologna, Italy
cnardini@deis.unibo.it

Luca Benini
University of Bologna
Bologna, Italy
lbenini@deis.unibo.it

Giovanni De Micheli
Stanford University
Stanford, USA
nanni@stanford.edu

Abstract

Clustering has been one of the most popular methods to discover useful biological insights from DNA microarray. An interesting paradigm is simultaneous clustering of both genes and experiments. This "biclustering" paradigm aims at discovering clusters that consist of a subset of the genes showing a coherent expression pattern over a subset of conditions. The pClustering approach is a technique that belongs to this paradigm. Despite many theoretical advantages, this technique has been rarely applied to actual gene expression data analysis. Possible reasons include the worst-case complexity of the clustering algorithm and the difficulty in interpreting clustering results. In this paper, we propose an enhanced framework for performing pClustering on actual gene expression analysis. Our new framework includes an effective data preparation method, highly scalable clustering strategies, and an intuitive result interpretation scheme. The experimental result confirms the effectiveness of our approach.

1. Introduction

The invention of DNA microarray spurred numerous efforts to acquire relative mRNA expression information from complex cellular systems [8]. Clustering has been one of the most popular among such efforts to discover useful biological insights from gene expression data [4, 7], and many novel clustering techniques have been proposed. An interesting paradigm is simultaneous clustering of genes and experimental conditions [3, 5, 6, 10, 11, 14]. The common objective is to discover clusters represented by a subset of the genes showing a coherent expression pattern over a subset of conditions. This paradigm is often referred to as *biclustering*, and such clusters are called *biclusters*. The concept of bicluster is common and intuitive, but its formal definition varies, depending on the measure of coherence and the clustering strategies.

The biclustering paradigm is more appropriate for the analysis of large-scale data than the traditional clustering methods for several reasons. First, the biclustering approach can better cope with the curse of dimensionality [4], which is frequently encountered in the analysis of high-dimensional data. In addition, biclustering is biologically more compatible with our understanding of cellular processes: we expect subsets of genes to be co-regulated and co-expressed under certain experimental conditions, but to behave almost independently under other conditions [3]. Discovering such local expression patterns may be the key to uncovering many genetic pathways that are otherwise not apparent.

The pClustering technique [14] belongs to this biclustering paradigm, and has many useful properties, compared with the well-known δ-biclustering method [5]. By measuring the coherence with finer granularity, the biclusters discovered by pClustering, or *pClusters*, are more homogeneous than δ-biclusters. Subclusters of a pCluster are also pClusters, whereas subclusters of a δ-biclusters are not necessarily a δ-bicluster. This property is more intuitive and convenient for designing efficient algorithms. In addition, the pClustering approach can find multiple clusters simultaneously, detects overlapping clusters better, and is more resilient to outliers [14].

Despite these theoretical advantages, it has been impractical to perform pClustering on actual gene expression data. First, the pioneering pClustering algorithm [14] depends on a nearly exhaustive procedure, and its worst-case complexity is exponential to the data size. Second, the original pClustering algorithm may fail to discover some important patterns existing in expression data. This is because the original algorithm is outperformed by an alternative algorithm presented in this paper, not only with respect to the response time, but also in terms of the number of pClusters that can be discovered. Third, pClustering can detect either *translation* (addition by a constant) or *scaling* (multiplication by a constant) patterns, but not both simultaneously. In actual expression data, the two patterns can coexist. Finally, it is hard to decide which pClusters are more

IEEE COMPUTER SOCIETY

valuable than others, since all pClusters are defined to be equally valid.

Our objective is to enhance the original pClustering method so that we can apply it to actual gene expression data analysis. Our contributions include the following.

- The enhanced pClustering algorithm EPC-EXACT that is more scalable than the original and can find *all* pClusters existing in a given data set.
- The polynomial-time pClustering algorithm EPC-POLY that is applicable under a certain condition; a polynomial-time algorithm to test the condition.
- A technique that can divide the huge expression data into subsets of manageable sizes, without compromising any global pCluster discovery. This method is useful to perform pClustering of very large-scale data, as well as to quickly produce representative pClusters without generating all.
- A data preparation method that helps our algorithm to detect complicated expression patterns more effectively. Also, a scoring scheme to help the users to interpret pClustering results.

2. Preliminaries

We assume the reader is familiar with the microarray technologies [7]. Let U_G be the set of genes and U_E be the set of experiments we are monitoring. Given a gene expression data matrix $D \in \mathbb{R}^{|U_G| \times |U_E|}$, let O be a subset of rows in D ($O \subseteq U_G$), and let T be a subset of columns ($T \subseteq U_E$). The pair $C = (O, T)$ denotes a submatrix of D.

2.1. Overview of pClustering

The pClustering approach [14] uses a coherence measure called *pScore*, in order to define *pCluster*, or the bicluster in the pClustering context.

Definition 1 *Given $g_1, g_2 \in O$ and $e_1, e_2 \in T$, the pScore of the 2×2 matrix is defined as*

$$pScore\left(\begin{bmatrix} v_{g_1e_1} & v_{g_1e_2} \\ v_{g_2e_1} & v_{g_2e_2} \end{bmatrix}\right) = |(v_{g_1e_1} - v_{g_1e_2}) - (v_{g_2e_1} - v_{g_2e_2})|$$

where $v_{g_ie_j}$ is the expression level of gene g_i in experiment e_j. Pair (O, T) forms a pCluster if, for any 2×2 submatrix X in (O, T), we have $pScore(X) \leq \delta$ for some $\delta \geq 0$. A maximal *pCluster is one that is not a subcluster of other pClusters.*

The problem of pClustering can be stated as follows. Given: (1) δ, a cluster threshold, (2) M_G, a minimal number of rows, and (3) M_E, a minimal number of columns, the task of pClustering is to find all pairs (O, T) such that (O, T) is a maximal pCluster according to Definition 1, and $|O| \geq M_G, |T| \geq M_E$.

2.2. Pairwise maximal pClusters

The pClustering problem is in general intractable, but we can find *all* maximal pClusters in a $2 \times n$ matrix in $O(nlogn)$ time [14]. Based upon this observation, pClustering starts with finding all the pairwise maximal pClusters in data, and then derive other pClusters from the pairwise maximal pClusters.

Definition 2 (Pairwise pCluster and π_E) *Assume $C = (\{g_i, g_j\}, T)$ is a $2 \times |T|$ pCluster. We call C pairwise pCluster of two genes g_i, g_j. The set of experiments T, is denoted as $\pi_E(g_i, g_j)$, if there does not exist $T' \supset T$ such that $(\{g_i, g_j\}, T')$ is also a $2 \times |T'|$ pCluster. That is, $\pi_E(g_i, g_j)$ is the set of experiments in a pairwise maximal pCluster.*

Each π_E and π_G usually contain the elements that do not contribute to the derivation of other clusters. The process of *pruning* [14] is to remove those elements.

2.3. Pairwise cluster tables (PCTs)

There can be multiple $\pi_E(g_i, g_j)$, given a pair of two distinct genes g_i, g_j, and so we define two more concepts to collectively refer to multiple π_E conveniently.

Definition 3 (Π_E and PCT_{gene}) *Given a pair of distinct genes g_i, g_j, we denote a set of all $\pi_E(g_i, g_j)$ as $\Pi_E(g_i, g_j)$. PCT_{gene} is a set of all possible π_E in a given data matrix.*

By switching the roles of genes and experiments, $\pi_G(e_i, e_j)$, $\Pi_G(e_i, e_j)$, and PCT_{exp} are defined *mutatis mutandis*. We refer the interested to [14] for the details about how to generate a PCT. There is no exponential growth of a PCT with regard to the data matrix size.

Figure 1(a) shows a gene expression matrix $D \in \mathbb{R}^{5 \times 5}$. (For simplicity, integer values were used). There exist 4 maximal pClusters on the data, with the parameters of $(M_G, M_E, \delta) = (3, 2, 1)$. The PCT_{gene} and PCT_{exp} derived from the data are also shown in the figure. The reader can verify $\pi_G(e_0, e_4) = \{g_0, g_1, g_3\}$, $\Pi_E(g_1, g_2) = \{\{e_0, e_4\}, \{e_0, e_2\}\}$, and $PCT_{exp} = \{\{g_0, g_1, g_2, g_3\}, \{g_0, g_1, g_3\}, \{g_0, g_1, g_3, g_4\}, \{g_1, g_3, g_4\}\}$.

3. The enhanced pClustering algorithm

Our approach is to classify the pClustering problem into three categories, according to a certain property of PCTs, and then to employ a different clustering strategy for each category.

IEEE COMPUTER SOCIETY

	e_0	e_1	e_2	e_3	e_4
g_0	1	3	2	5	4
g_1	2	5	4	3	6
g_2	2	2	3	6	7
g_3	3	2	5	2	6
g_4	4	9	3	1	4

(a) A data matrix D

#	O	T
0	$\{g_0, g_1, g_3\}$	$\{e_0, e_2, e_4\}$
1	$\{g_0, g_1, g_2, g_3\}$	$\{e_0, e_2\}$
2	$\{g_0, g_1, g_3, g_4\}$	$\{e_2, e_4\}$
3	$\{g_1, g_3, g_4\}$	$\{e_3, e_4\}$

(b) Maximal pClusters on D

Genes	Π_E
(g_0, g_1)	$\{e_0, e_1, e_2, e_4\}$
(g_0, g_2)	$\{e_0, e_2, e_3\}$
(g_0, g_3)	$\{e_0, e_2, e_4\}$
(g_0, g_4)	$\{e_2, e_4\}$
(g_1, g_2)	$\{e_0, e_4\}, \{e_0, e_2\}$
(g_1, g_3)	$\{e_0, e_2, e_4\}, \{e_3, e_4\}$
(g_1, g_4)	$\{e_2, e_3, e_4\}$
(g_2, g_3)	$\{e_0, e_2\}, \{e_0, e_1\}, \{e_1, e_4\}$
(g_3, g_4)	$\{e_2, e_3, e_4\}$

(c) PCT_{gene}

Experiments	Π_G
(e_0, e_2)	$\{g_0, g_1, g_2, g_3\}$
(e_0, e_4)	$\{g_0, g_1, g_3\}$
(e_2, e_4)	$\{g_0, g_1, g_3, g_4\}$
(e_3, e_4)	$\{g_1, g_3, g_4\}$

(d) PCT_{exp}

Figure 1. A data matrix, maximal pClusters and pairwise cluster tables (PCTs).

3.1. Problem classification

We first introduce the $\otimes$ operator and the concept of *well-shaped* PCTs.

Definition 4 ($\otimes$ operator) *Given two set of subsets A and B, $A \otimes B = \{\eta | \eta = \alpha \cap \beta,\ \forall \alpha \in A$ and $\forall \beta \in B\}$.*

Definition 5 *The table PCT_{gene} is well-shaped if, $PCT_{gene} \supset \{\pi | \pi \in \Pi_E(g_i, g_j) \otimes \Pi_E(g_l, g_m)$, and $|\pi| \geq M_E\}$, for any two distinct gene set pairs (g_i, g_j) and (g_l, g_m) in PCT_{gene}. We define well-shaped PCT_{exp} similarly.*

For instance, PCT_{gene} in Figure 1(c) is well-shaped, because the result of the $\otimes$ operation on any pair of Π_E is a subset of PCT_{gene}, ignoring the sets of cardinality less than $M_E = 2$.

We can classify the pClustering problem into three categories, according to the well-shapedness of PCTs.

Type 1 Both PCT_{exp} and PCT_{gene} are well-shaped. We use the EPC-POLY algorithm presented in Section 3.2.

Type 2 Either PCT_{exp} or PCT_{gene} is well-shaped (not both). Section 3.3.3 explains how to solve Type 2 problems as a special case of Type 3 problems.

Type 3 No PCT is well-shaped. We use the EPC-EXACT algorithm in Section 3.3.

We can test in polynomial time if a PCT is well-shaped, since there is no exponential growth of PCTs and the relation in Definition 5 can be verified in polynomial time with respect to the size of D.

3.2. The EPC-POLY algorithm

The well-shaped PCT_{gene} has a very desirable property: for any arbitrary maximal bicluster (O, T), its experiment set T exists in PCT_{gene}. Thus, we can discover any T, by probing PCT_{gene}. Suppose $O = \{g_1, g_2, g_3, \ldots, g_N\}$. By Property 1 in Section 3.3,

$$\begin{aligned} T &\in \bigotimes_{\forall (g_i, g_j) \in O} \Pi_E(g_i, g_j) \\ &= \underbrace{\Pi_E(g_1, g_2) \otimes \Pi_E(g_1, g_3)}_{\triangleq \Pi_1 \subset PCT_{gene}} \otimes \Pi_E(g_1, g_4) \cdots \\ &= \underbrace{\Pi_1 \otimes \Pi_E(g_1, g_4)}_{\triangleq \Pi_2 \subset PCT_{gene}} \otimes \Pi_E(g_1, g_5) \cdots \\ &= \cdots \\ &= \Pi_{\frac{N(N-1)}{2} - 1} \subset PCT_{gene}. \end{aligned}$$

$\Pi_1, \Pi_2, \ldots, \Pi_{\frac{N(N-1)}{2}-1} \subset PCT_{gene}$, since PCT_{gene} is well-shaped. Therefore, T can be found in PCT_{gene} if it is well-shaped. The well-shaped PCT_{exp} has the same advantage for discovering O.

If both PCTs are well-shaped, as is the case for Type 1 problems, the building blocks for any maximal pCluster (O, T) already exist in PCT_{exp} and PCT_{gene}, respectively. We only need to pair them appropriately. Figure 2 gives the outline of the EPC-POLY algorithm.

```
input  : well-shaped $PCT_{gene}$ and $PCT_{exp}$
output : all maximal pClusters
for each $\pi_G$ in $PCT_{exp}$ do
    for each $\pi_E$ in $PCT_{gene}$ do
        if $(\pi_G, \pi_E)$ forms a pCluster then
            report $(\pi_G, \pi_E)$;
Remove nonmaximal pClusters;
```

Figure 2. The EPC-POLY algorithm

3.3. The EPC-EXACT algorithm

Not all PCTs are well-shaped in general, and we introduce a property that holds in any maximal pCluster, regardless of the well-shapedness of a PCT.

Property 1 *Given a maximal pCluster $C = (O, T)$,*

$$T \in \bigotimes_{\forall (g_i, g_j) \in O} \Pi_E(g_i, g_j), \textit{ and } O \in \bigotimes_{\forall (e_i, e_j) \in T} \Pi_G(e_i, e_j).$$

Example 1 *For the cluster 0 in Figure 1(b), $O = \{g_0, g_1, g_3\}$ and $T = \{e_0, e_2, e_4\}$. $T \in \Pi_E(g_0, g_1) \otimes \Pi_E(g_0, g_3) \otimes \Pi_E(g_1, g_3) = \{\{e_0, e_2, e_4\}, \{e_4\}\}$.*

For simplicity, we use the second relation only in the sequel. Discovering pClusters through this relation is a two-step process: we first find T, and then perform $\otimes$ operations in order to determine O. For T, it is enough to try each π_E, and its subsets not smaller than M_E, since $T \subseteq \pi_E$ and $|T| \geq M_E$. In order to avoid excessive and repetitive enumeration of the subsets of π_E, we keep track of them using the *trie* data structure [1]. Since there frequently exist overlaps among π_E, the trie also provides an effect of compaction.

3.3.1. Generating and pruning the Trie

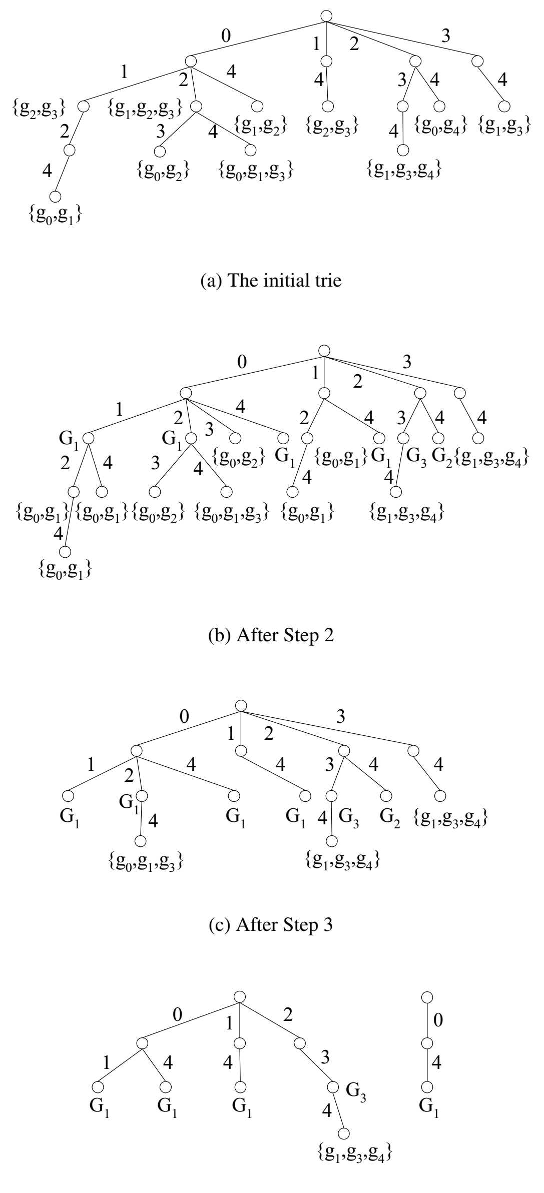
In a trie, each path from the root to a leaf corresponds to one word or character string in the represented set.

(Step 1) For each $\pi_E(g_i, g_j)$ in PCT_{gene}, we first sort the elements in $\pi_E(g_i, g_j)$ in an ascending order. Then insert $\{g_i, g_j\}$ into the node whose path is specified by the ordered elements. In Figure 3(a), $\{g_0, g_1\}$ of $\pi_E(g_0, g_1) = \{e_0, e_1, e_2, e_4\}$ is inserted into the leftmost leaf by following the path *"0,1,2,4"*. The nodes whose level (the level of the root is 1) is less than M_E are empty, since no π_E has less elements than M_E.

(Step 2) We expand the trie according to the following observation: if two genes $\{g_i, g_j\}$ form a pCluster with π_E, then they must form a pCluster with subsets of π_E, too. Thus, traversing the trie in *post-order*, we distribute all the gene elements of a node n to other nodes whose path is one shorter than the path to n, but not less than M_E. (Figure 3(b))

(Step 3) We now reduce the size of the trie by removing those nodes that have less than M_G genes. We can do this step efficiently by the *pre-order* traversal of the trie. This is because the genes a node has are always the same as or more than those its children have. Thus, if a node has less than M_G genes, then none of its children can have more. For this reason, we can safely remove the entire subtree whose root is located at that node. (Figure 3(c))

(Step 4) Now we can consider the genes in each node and the experiments represented by the path to that node as a candidate cluster. In a certain node the genes and exper-



(a) The initial trie

(b) After Step 2

(c) After Step 3

(d) After Step 4 (e)

Figure 3. An example of the trie construction, expansion, and pruning. ($G_1 = \{g_0, g_1, g_2, g_3\}$, $G_2 = \{g_0, g_1, g_3, g_4\}$, $G_3 = \{g_0, g_1, g_2, g_3, g_4\}$.)

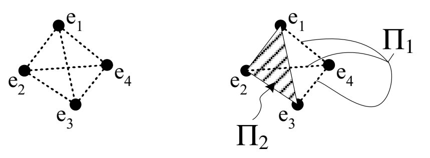


Figure 4. Example for reducing $\otimes$ operations.

iments can be already forming a pCluster. We collect the pCluster and remove the node if it is a leaf. (Figure 3(d))

3.3.2. Reducing total number of $\otimes$ operations It is important to reduce the number of $\otimes$ operations as much as possible, since they are computationally expensive. Our approach is similar to dynamic programming, where previously calculated results are exploited.

In the example in Figure 4, the node e_i and the edge (e_i, e_j) in the graph represents experiment e_i and $\Pi_G(e_i, e_j)$, respectively. The left figure represents the case where previous results are not saved. With n nodes in the graph, we need to look at all $\binom{n}{2}$ edges, resulting in $\binom{n}{2} - 1$ $\otimes$ operations. In contrast, by storing the previous results as in the right figure, we can reduce the number of edges to look at to $(n - 1)$. Let Π_1 denote the intermediate result obtained by performing $(n - 2)$ $\otimes$ operations among these edges. Π_2 or the shaded triangle represents the stored result. The final answer can be obtained by $\Pi_1 \otimes \Pi_2$. That is, we now need only $(n - 2 + 1) = (n - 1)$ $\otimes$ operations. We apply this idea to the trie as follows.

(Step 5) We calculate the intermediate data corresponding to Π_1 in Figure 4 for each node, traversing the trie in *pre-order*. If this result at a certain node is empty, we remove the entire subtree rooted at that node, because applying further $\otimes$ to a null set is meaningless. (Figure 3(e))

(Step 6) As traversing the trie in *pre-order*, we perform the operation corresponding to $P = \Pi_1 \otimes \Pi_2$ in Figure 4. Π_1 corresponds to the result stored in the parent node. If P is empty for a node, we prune the entire subtree rooted at that node. Otherwise, we collect pClusters (denoted by elements in P and the path to that node), if any, and store P for the later use by the children.

(Step 7) We remove any nonmaximal pClusters, if any, from the collection of pClusters discovered so far, and report the remaining pClusters.

3.3.3. Clustering of type 2 problems If PCT_{gene} is well-shaped and PCT_{exp} is not, we run the same EPC-EXACT algorithm, only skipping Step 2. This is because we do not need to consider *subsets* of π_E at all, if PCT_{gene} is well-shaped. With no expansion of the trie, this modified algorithm is faster than EPC-EXACT. In the opposite case where only PCT_{exp} is well-shaped, we insert π_G rather than π_E into the trie, and follow the same procedure.

3.4. Remarks

The original pClustering algorithm in [14] is a special case of the EPC-EXACT algorithm. The original algorithm is for the case where $|\Pi_G(e_i, e_j)| = 1$ for any e_i, e_j. Although the EPC-EXACT algorithm has the worst-case complexity above polynomial, its running time on typical benchmarks is practical. The worst-case complexity of the EPC-POLY algorithm depends on the size of PCTs, which are polynomial with respect to the data size even in worst-case.

4. Additional enhancements

4.1. Dividing data matrices

We introduce a technique to divide the whole expression data matrix into submatrices, without compromising any global pCluster discovery. The resulting submatrices will be small enough to apply EPC-EXACT or EPC-POLY, even if the original data matrix is so huge that the algorithms are not applicable. We can discover all pClusters as before, if we run the algorithms on every submatrix. Otherwise, we can quickly produce a representative sample of pClusters by focusing on a well-distributed set of submatrices.

The basic idea is to divide the data matrix into submatrices specified by (π_G, U_E). The motivation is that for any pCluster (O, T), O is always a subset of a certain π_G. If we examine every π_G, we can discover all pClusters. Moreover, this division can be done very quickly, since π_G generation is trivial even for large-scale data. In most gene expression data, $\pi_G \ll |U_G|$ and $|U_E| \ll |U_G|$, so the size of each submatrix is manageable.

If the objective is to quickly generate a sample of pClusters rather than discover all, we can divide the matrix according to the algorithm.

(1) We first construct PCT_{exp} from the data matrix D, as shown in Figure 5. Each **a**-**f** in the PCT_{exp} represents a $|\pi_G| \times 2$ submatrix corresponding to $(\pi_G, \{e_i, e_j\})$. (If we want to find all pClusters, we run EPC-EXACT or EPC-POLY on each (π_G, U_E), and finish.)

(2) We choose n π_G from the PCT_{exp} in such a way that they cover U_G. For each selected $\pi_G(e_i, e_j)$, we form a $|\pi_G| \times 2$ submatrix corresponding to $(\pi_G, \{e_i, e_j\})$ and call it a *seed*. In Figure 5, we assumed **a** and **c** were selected as seeds, because the union of π_G in **a** and **c** is equal to U_G.

(3) For each $\pi_G(e_l, e_m)$ not selected as a seed, we compare it with the gene set π'_G in each seed and mea-

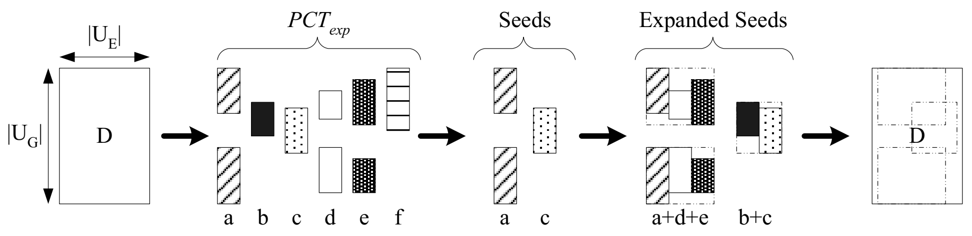


Figure 5. Dividing a large data matrix into submatrices of manageable sizes.

sure similarity between them. If π_G and π'_G are determined to be close enough by a certain criterion (such as "overlap greater than 95%"), we merge that π_G with the seed. That is, we expand the seed vertically by taking a union of π_G and π'_G, as well as horizontally by taking a union of the experiment sets. If that π_G is not similar to any of the π'_G in the seeds, then we remove it. This heuristic does not affect the quality of clusters forming on the expanded seeds.

(4) We perform EPC-EXACT or EPC-POLY on each of the n expanded seeds.

4.2. Data preparation

The translation or scaling patterns that can be discovered by the original pScore model would be quite limited, as mentioned in Section 1. This limitation can be alleviated by a proper preprocessing. One obvious way is normalization of the data matrix. If we do not want the piecewise derivative of a pattern curve to be changed, which can happen in normalization, we can perform a simple signal range transformation.

We replace the original value x in the data with a certain $f(x)$. We observe that $f(x) = sign(x)\sqrt{|x|}$ and its variants work well. By such $f(x)$, x is made bigger under a certain value t and smaller above it. On average, the chance of being dropped from further considerations increases for subthreshold or "small" x, whereas it decreases for "large" x. This transformation is statistically meaningful [9] and also conforms with the biological observation that the correlation between highly expressed genes are more important than that between the vaguely expressed [6].

4.3. Cluster interpretation

Sometimes a large number of pClusters can be reported. We can reduce the number by adjusting parameters. Generating many pClusters is often inevitable, however, if we use the exact pClustering algorithms in order to monitor all pClusters. This large number of pClusters makes it difficult to interpret the results, since all pClusters are equally valid by definition, unless discriminated by other measures. In order to pay more attention to interesting results, we propose to rank pClusters according to a certain measure that can summarize the degree of coherence existing in clusters. We use the *mean square residue (MSR)* score, which is the measure of the coherence in δ-biclustering [5]. Using the MSR score also makes it possible to compare the result from our enhanced pClustering and that from δ-biclustering, as shown in Section 5.

5. Experimental Results

We implemented both our enhanced pClustering algorithm and the original algorithm in [14] for comparison. We used the synthetic and the real gene expression data listed in Table 1. The synthetic data were generated by the method

Data ID	$\|U_G\|$	$\|U_E\|$	Origin	Ref.
D_{iK}	i × 1,000	30	Synthetic	[14]
D_{tumor}, D^+_{tumor}	1,000	16	Liver	[13]
D_{yeast}	2,884	17	Yeast	[5, 12]
D_{lymp}	4,026	96	B-cell	[2]

Table 1. Data sets for the experiment.

introduced in [14]. For the real data, we used the expression data from yeast [5, 12], human B-cell lymphoma [2], and human liver cancer [13]. $|U_G|$ and $|U_E|$ denote the row and column size of the data matrix, respectively. In the synthetic data set, D_{8K}, for instance, has 8,000 rows. For each D_{iK}, we prepared 10 different data sets. D^+_{tumor} is the preprocessed version of D_{tumor} by $f(x) = sign(x)\sqrt{|x|}$.

All the experiments were conducted on a 900 MHz SPARC-III+ with 1 GB RAM, except for the one in Table 2, which was ran on a 3.06 GHz Linux machine with 4 GB RAM.

5.1. Comparison with the original algorithm

We first experimented our algorithm on the synthetic data sets to show its correctness and performance improvements. Figure 6 shows the average running time (in seconds) on synthetic data set D_{iK}, with

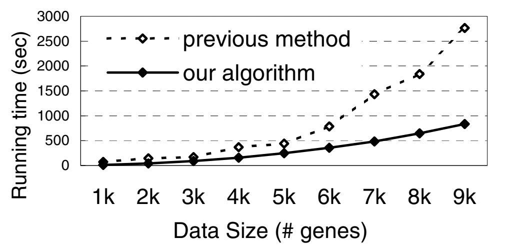


Figure 6. Running time comparison.

$i = 1000, 2000, \ldots, 9000$. We noticed a larger speedup for bigger data matrices. We also verified that our algorithm can discover all clusters inserted in advance to the synthetic data sets.

We then applied the algorithm to D_{yeast} to show that our approach can discover more pClusters. Table 2 presents the results. In the table, #pC and #Add mean the total number of pClusters reported and that of pClusters found by our method only, respectively. The data set was D_{yeast}. The two algorithms gave the same result for some cases. However, more pClusters were found by our algorithm when the size of PCT is relatively large and thus most Π_G are expected to have multiple elements (Section 3.4). We also confirmed the validity of the additional pClusters our method discovered.

M_G	M_E	δ	RT (s)	|PCT|	#pC	#Add.
38	8	1	31.21	12,915	5	0
20	10	4	49.62	3,447	22	0
40	7	1	42.30	39,777	106	0
42	7	2	50.69	37,969	67	0
35	7	2	75.29	43,559	404	165
33	7	2	131.69	49,223	645	362
32	7	2	158.74	51,085	807	433
40	6	1	113.71	136,265	1,204	1,189
42	6	2	174.60	127,876	910	891
40	5	1	172.87	345,413	4,095	4,086

Table 2. The number of pClusters discovered.

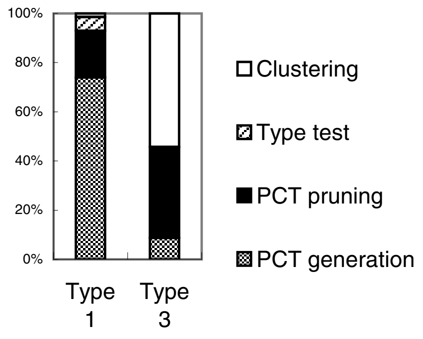
5.2. Additional experiments

In the following experiments, the original pCluster algorithm could not respond in reasonable time, and thus we present the results from our algorithms only.

5.2.1. Distribution of problem type We observed that Type 1 and Type 3 were more common than Type 2. An interesting observation was, the problem type can vary depending upon the choice of (M_G, M_E, δ), even for the same data set. When we used the parameters generating many overlaps, for instance, the problem normally belonged to Type 3. In contrast, when we chose such parameters that resulted in few overlaps, the problem was mostly Type 1. The synthetic data sets D_{iK} all belonged to Type 1, regardless of parameters. This is because the clusters were inserted, by design, without overlaps. The original pClustering was applicable to large-scale Type 1 problems such as D_{iK}, but not to most Type 2 or 3 problems.

M_G	M_E	δ	RT_1(s)	RT_3(s)	TT(s)
40	10	5	647.8	2068.29	2.34
30	10	5	658.2	2341.64	2.76
40	8	5	667.9	1627.91	8.64
30	8	4	673.7	1664.50	8.79
30	7	5	860.8	1693.20	4.01
20	10	3	1230.3	2052.15	5.49

(a) Running time



(b) Running time breakdown

Figure 7. Effects of problem type.

5.2.2. EPC-EXACT versus EPC-POLY We also tested our algorithm on D_{lymph} to show the effect of problem type on performance. We tried many different sets of parameter triplet, some of which are shown in Figure 7. The first table gives the running time with different parameters on Type 1 problems. The data set was D_{lymph} and RT_1, RT_3, and TT stand for the runtime for Type 1 algorithm, Type 3 algorithm, and the test algorithm to figure out the problem type, respectively. In this particular data set, we observed that most test cases belong to Type 1. To show the effectiveness of our Type 1 algorithm, we also listed running time of Type 3 algorithm. We observe that the time to test the problem type is negligible.

The plot in Figure 7(b) shows the running time breakdown according to the problem type. The PCT generation step dominates in Type 1 problems. The clustering step dominates in Type 3 problems. The D_{iK} was used for Type 1 and D_{yeast} for Type 3.

ID	row	col	MSR
86	28	5	115.6
72	18	7	134.5
98	9	7	161.5
90	28	7	186.2
80	23	8	217.7
83	9	8	220.9
84	11	10	249.9

(a) δ-biclusters [5]

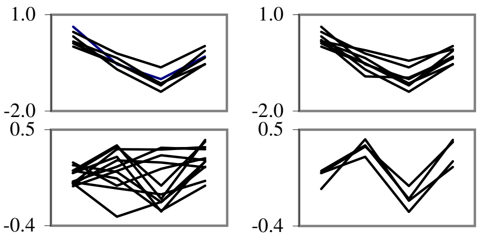
ID	row	col	MSR
228	28	5	44.0
152	30	5	46.6
185	29	5	47.3
207	28	5	58.9
0	10	7	43.9
12	10	7	47.8
1	10	7	51.1

(b) pClusters

Table 3. MSR score comparison.

5.2.3. Comparison with δ-biclusters We compared pClusters and δ-biclusters, in terms of the mean residue square (MSR) score. The details are listed in Table 3. Among the 100 δ-biclusters on D_{yeast} reported in [5], we selected 7 δ-biclusters of reasonable size and relatively low MSR scores, as listed in Table 4(a). For comparison, we generated pClusters of similar sizes, and ranked them according to their MSR scores, as shown in Figure 4(b). The first 4 were generated with $(M_G, M_E, \delta) = (28, 5, 51)$. We show only the pClusters that have 4 lowest MSR scores. The other entries are the pClusters that have 3 lowest MSR scores among the pClusters produced with $(M_G, M_E, \delta) = (10, 7, 48)$. We observed that the MSR scores of the pClusters are consistently lower than the δ-biclusters of similar size, meaning that more coherently regulated gene expression patterns were detected by the pClusters.

5.2.4. Effect of pre-processing We ran our enhanced pClustering algorithm on D_{tumor} and D^+_{tumor} and compared the result. Figure 8(a) shows some pClusters discovered on D_{tumor} with $\delta = 0.7$. When the signal range was quite bigger than δ (the upper plot in the figure), the cluster formation was reasonable. In contrast, when the signal range was comparable to δ (the lower plot), the cluster was noisy. Figure 8(b) presents some pClusters discovered on D^+_{tumor} with $\delta = \sqrt{0.7}$. The pClusters found on D^+_{tumor} were less affected by the signal range.



(a) No preprocessing (b) With processing

Figure 8. Effect of pre-processing.

6. Conclusions

In this work, we proposed a suite of new algorithms to perform pClustering on large-scale expression data. The experimental results confirmed that our enhanced pClustering approach is more scalable, finds more pClusters, and detects more complicated patterns than the original pClustering technique. Leveraged by our new framework, we hope more applications of pClustering to gene expression data analysis will come out.

Acknowledgments

The authors would like to thank Stefano Volinia and Massimo Negrini at University of Ferrara for providing us with the human liver cancer data. This work was supported by a grant of Jerry Yang and Akiko Yamazaki.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. McGraw-Hill, Addison-Wesley, 1983.
[2] A. Alizadeh and et all. Distinct types of diffuse large b-cell lymphoma identified by gene-expression profiling. *Nature*, 4051:503–511, 2000.
[3] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: The order-preserving submatrix problem. In *Proceedings of RECOMB*, 2002.
[4] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.
[5] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proceedings of ISMB*, pages 93–103, 2000.
[6] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. *Proc. Natl. Acad. Sci USA*, 94:12079–12084, 2000.
[7] I. S. Kohane, A. T. Kho, and A. J. Butte. *Microarrays for an Integrative Genomics*. The MIT Press, Cambridge, Massachusetts, 2003.
[8] S. Raychaundhuri, P. D. Sutphin, J. T. Chang, and R. B. Altman. Basic microarray analysis: grouping and feature reduction. *Trends in Biotechnology*, 19(5):189–193, May 2001.
[9] B. Rosner. *Fundamentals of Biostatistics*. Duxbury, Pacific Grove, California, fifth edition, 2000.
[10] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, 17(Suppl 1):S243–52, 2001.
[11] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. In *Proceedings of ISMB*, 2002.
[12] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22:281–285, 1999.
[13] S. Volinia and M. Negrini. (personal communication). Univerity of Ferrara – International Cancer Center, Rovigo, Italy.
[14] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *Proceedings of ACM SIGMOD*, pages 394–405, 2002.