Identification of Shifting Regulatory Modules in Time Series Gene Expression Data Using a Linear Time Biclustering Algorithm

Tustanah Phukhachee and Songrit Maneewongvatana

Abstract— **Since standard biclustering problem was defined, the problem is known to be NP-hard. However in analyzing time series expression data, we can restrict the problem with the trait of data that represented the contiguous columns, which corresponded to coherent expression patterns. With this restriction included, the problem considered to be tractable problem. We propose an algorithm to find and report all maximal contiguous column coherent biclusters with the shifting input included in time linear within the size of expression matrix multiplied by the size of the shifting window.**

Index Terms—**Biclustering, regulatory modules, shifting input data, suffix tree, time series gene expression data.**

I. Introduction

Clustering microarray data is one of challenging problems nowadays. Since the DNA chips techniques enable simultaneous measurements of the expression level of a wide range of genes for given experiment conditions [1]. These wide ranges of the data and complexity of clustering microarray data problem make the problem to be difficult.

Biclustering is the technique that simultaneously clusters the microarray data on both genes and conditions. The data in each cluster exhibit highly correlated behaviors between the subgroups of genes and conditions. It has showed many advantages in identification the local expression patterns and has been extensively studied and surveyed [2]-[4]. Biclustering problem has many approaches proposed to date. Most of the biclustering approaches presented are heuristic and thus do not guarantee to find the optimal solution. Some other cases used exhaustive search which imposed on the size of the biclusters in order to obtain reasonable runtimes. The dealing with exact value of original expression data matrix is one difficulty of biclustering problems. And finding coherent behaviors regardless of the exact values are becomes great interest. These circumstances lead to founding of many new methods based on discrete matrix [5]-[18]. Unfortunately, these versions remain to be NP-hard problem.

Although the problem still remains to be NP-hard, there exist some restrictions to the biclustering problem which lead it to be a tractable problem. For example, if the expression data are organized in a way that expression level of various snapshots of the same condition are represented as group of time-sorted contiguous columns and biclusters are limited to contiguous snapshots. This study focuses on such setting, which uses to identify coherent expression patterns shared by a group of genes in consecutive time points.

We are also interested in finding the time shifted coherent biclusters. Time shifted coherent bicluster is a bicluster where each gene in the cluster exhibits the same coherent pattern over different set of contiguous time series columns. By observing time shifted coherent patterns, it discovers more related and hidden patterns than the standard techniques used nowadays.

In this work, we propose an algorithm to find and report all maximal shifting contiguous column coherent biclusters (SCCC-Biclusters) in the linear time and in the size of expression matrix multiplied by the size of the shifting window. Maximal SCCC-Bicluster is a SCCC-Bicluster which no existed SCCC-Biclusters that can be the superset of it. Our algorithm is based on contiguous column coherent biclusters (CCC-Biclusters) [19] and is improved by considering the shifting of the input expression data in contiguous time during the suffix tree creation. With the shifted data considered in our work, the result constructed suffix tree we get are more complicated than the original problem. This complicated suffix tree lead to finding the insignificant biclusters. In order to deal with the insignificant biclusters from the result, we introduce the post process method to remove the insignificant biclusters derived from the suffix tree.

The content in this paper is organized as follows: in Section II, we survey related work. In Section III, notations used throughout this paper are defined. In Section IV, the shifting contiguous column coherent biclusters algorithm is proposed. Finally, we concluded our work in Section V.

II. Related Work

At present, there are a large number of biclustering algorithms which were proposed to solve the general case of biclustering [3], [4]. We can group biclustering algorithms into 3 main groups as follows: heuristic algorithms, exhaustive algorithms, and condition-based algorithms.

The large majority of biclustering algorithms use heuristic approaches to identify biclusters [3]. The Coupled Two-Way Clustering (CTWC) [20] which uses only subsets of rows or columns that are identified as stable clusters in previous clustering iteration are candidates for the next iteration. This heuristic leads to avoid all possible combinations which reduce search time of the algorithm with the cost of accuracy of the algorithm.

Manuscript received December 9, 2012; revised February 15, 2013.

Tustanah Phukhachee and Songrit Maneewongvatana are with the Computer Engineering Department, Faculty of Engineering, King Mongkut's University of Technology Thonburi, Bangkok, Thailand (e-mail: s54450015@st.kmutt.ac.th, songrit@cpe.kmutt.ac.th).

Cheng and Church [2] introduced the first biclustering algorithm applied to gene expression data. This algorithm is also heuristic algorithm. The algorithm starts with iteratively removal of a row or a column that gives the maximum decrease of similarity score, mean squared residue, H, until no further decrease of H. Then the algorithm iteratively adds a row or a column which gives minimum increase of H back to construct the bigger similarity biclusters with H lower than some threshold, δ. Finally, it reports the bicluster and marks the newly found bicluster in order to find the next one. This heuristic method makes the algorithm very fast. However with this techniques and the marking make the discovery of highly overlapping biclusters unlikely, since elements of already identified biclusters have been marked by random noise [3].

On the contrary, the exhaustive algorithms such as brute force or exhaustive enumeration methods can find the optimal set of biclusters [3]. However, due to their high complexity, they can be applicable only when the input size is small [3]. This led the practical exhaustive algorithms to be condition- based algorithm instead.

Although there is some condition-based biclustering algorithm which addressed the consecutive columns, the first was purposed by Ji and Tan [6]. The exact complexity of this algorithm is hard to estimate by their description. Still this algorithm considered to be in linear time. Yet, there exist another proposal which addressed the problem of finding maximal contiguous column coherent biclusters (CCC-Biclusters) [19].

Madeira *et al.* [19] proposed the CCC-Biclusters algorithm. This algorithm can find and report all the maximal contiguous column coherent biclusters in linear time and in the size of the expression matrix. Although our context is based on this algorithm, however in our work we also proposed the improvement to find the maximal S-CCC Biclusters altogether. This will increase flexibilities and open the new path in finding the larger significant biclusters from the data matrix which will improve the efficiency and accuracy to the standard CCC-biclustering algorithm.

In fact, there existed another work of Madeira et al. [21] which addressed to the shifting time of gene expression data similar to our work as the extension of CCC-Biclusters called time-lagged activation. However in their work they used the exhaustive algorithm with each time lag vary from one to the size of column minus one. This makes the complexity of their algorithm to be at $O(|R|^2|C|^2)$, with |R| referring to size of rows and |C| referring to size of columns, it is thus not in linear time on the input size. In our work we use the shifting window size, w, and generate the shifting pattern of the input data to be temporally input data to be input of the algorithm which leads our algorithm to achieve linear time complexity, $O(|R||C|w)$.

III. Definitions

Our definitions of CCC-Biclusters and suffix tree are based on [19]. However we also include more definition about shifting data, shifting window size that relate to our work.

We denote gene expression matrix with $|R|$ rows and $|C|$ columns as M, where R, be the set of genes and C, be the set of its conditions. The expression of the genes i under condition j is represented by M_{ij}.

In this work, the input data used was already discretized to be in regulatory modules with 3 symbols as $\{D, N, U\}$ which refer to DownRegulated, NoChange, and UpRegulated respectively.

Our definitions used throughout this work are defined as follows:

Definition 1 (*bicluster and trivial bicluster*). Bicluster $B = (I, J)$ is the submatrix M_{ij} defined by $I \subseteq R$, subset of rows, and $J \subseteq C$, subset of columns. Trival bicluster is a bicluster with only 1 row and 1 column. Our goal is to identify the biclusters which exhibit coherent evolution.

Definition 2 (*CC-Bicluster*). CC-Bicluster M_{ij} is the bicluster which $M_{ij} = M_{lj}$ for all rows $i, l \in I$ in the same column $j \in J$. Finding the optimal set of maximal biclusters which satisfy this coherence property is known to be an NP-hard problem [22].

Definition 3 (*CCC-Bicluster*). CCC-Bicluster M_{ij} is subset of rows I, $I = \{i_1, i_2, i_2, ..., i_k\}$, $k \leq |R|$, and contiguous subset of columns J, $J = \{s, s+1, s+2, ..., f-2, f-1, f\}$, which $s \geq 1$, $f \leq |C|$, $M_{ij} = M_{lj}$ for all rows $i, l \in I$ and columns $j \in J$. In this work, we considered only the time series expression data which have contiguous column trait; therefore we can reduce the complexity of the problem to be tractable problem with contiguous column result.

- C1 C2 C3 C4 C1 C2 C3 C4
- G1 N D U D G1 N D U D
- G2 U D N N G1*1 * N D U
- G1*2 * * N D
- G2 U D N N
- G2*1 * U D N
- G2*2 * * U D
- (a) (b)

Fig. 1. (a) Example of original input expression matrix. (b) The input expression matrix with original expression data and shifted data which window size equal to 3.

Definition 4 (*Shifting window size and Shifting data*). Shifting window size, w, is the number of columns from the starting column (counting start column as 1) to the last column that the data in this column will compare the data with. Our algorithm will create the shifting data (if w is 2 or more) and insert it as the input data, we denoted this shifting window size as w. Fig. 1 illustrates the original expression matrix (w equal to 1) and its transformed form after including shifted data whose window size is equal to 3.

Definition 5 (*SCCC-Bicluster*). CCC-Bicluster M_{ij} is shifted CCC-Bicluster, SCCC-Bicluster, if the CCC-Bicluster is the result from the biclustering algorithm which considered all the shifting data within the specific shifting window size of input into its biclustering process.

Definition 6 (*row-maximal SCCC-Bicluster*). SCCC-Bicluster M_{ij} is row-maximal if we cannot add more rows to the subset of rows I of the bicluster and still maintain the coherence property of J in Definition 3.

Definition 7 (*left-maximal and right-maximal SCCC-Bicluster*). SCCC-Bicluster is left-maximal/ right-maximal if we cannot add more symbols to the beginning/end of its expression pattern (contiguous column) without changing its set of rows I.

Definition 8 (*maximal SCCC-Bicluster*). SCCC-Bicluster M_{ij} is maximal if no other SCCC-Bicluster exists that can be its superset, for all other SCCC-Bicluster M_{ST}, if $I \subseteq S$ and $J \subseteq T$ then $I = S$ and $J = T$. Hence maximal SCCC-Biclusters are the SCCC-Biclusters which are right, left, and row-maximal.

Definition 9 (*suffix tree and generalized suffix tree*). Suffix tree is a kind of rooted tree which encodes every suffix of a string to construct the tree. Suffix tree T for string S is a tree whose has $|S|$ edges and each edge is label with a part of string S. These parts represented each position of each symbol of S to the final symbols of $|S|$. The 3 main properties of suffix tree are as follows:

1) Each internal node except root node has at least two children.
2) Two edges which come out of the same node must not label with the same symbol.
3) The path to any leaf i must be exactly labeled as the symbol of string from position i to $|S|$.

Generalized suffix tree is suffix tree build from the set of strings instead of a single string like suffix tree.

Definition 10 (*string depth and string label*). String depth is the length of string to the specific position. String depth of node v in suffix tree T is the length of the symbols from the root node to node v in T we denote this as $P(v)$ and denote the path which contain all these symbols from the root to v as string label.

Definition 11 (*number of leaves*). Number of leaves is the number of leaves that come out of the internal node in the suffix tree. For each node v in suffix tree T we denote the number of leaves in the subtree with v as root node by $L(v)$.

Definition 12 (*suffix link*). For the node v in T with label $x\alpha$, where x is a single character and α is a string (possibly empty), if there is another node u with label α then there will be a link point from v to u, we define this link as suffix link. The special case is when α is empty then $x\alpha$ has a suffix point from v to the root.

Definition 13 (*MaxNode*). MaxNode is an internal node v of the suffix tree T which satisfies one of these following conditions:

1) Node v does not have incoming suffix links.
   Or
2) 2) Node v has only incoming suffix links from node u such that for every node u, $L(u) < L(v)$.

IV. Algorithms

A. Preprocessing Step

Our algorithm assumed that the gene expression input data has already been discretized. Therefore, first part of our work is to apply the alphabet transformation technique which was introduced by Madeira et al. [19] to our discrete input data.

In alphabet transformation process, for each string S in set of strings $\{S_1^o, ..., S_{|R|}^o\}$, we append each S by its position in the column, let $S_i^{'}$ be the symbol of the string S at column i, then $S_i^{'} = S_i^o i$.

We then append terminator symbols to each string S. This is required in order to follow the definition of suffix tree, when one suffix of S matches the prefix of another suffix of S we add a symbol to its end. This symbol must not appear anywhere else (usually symbol \$ was used). For each string $S^{'}$ in set of strings $\{S_1^{'}, ..., S_{|R|}^{'}\}$, we inserted the special symbols $\$x$ to the end of the string where x is the row of that string, $S_x^{"} = S_x^{'}\$x$. Therefore, our last column now is termination symbol and our columns size is increased by 1.

B. Shifting Data Process and Extend Input Data Matrix

To let our algorithm identify shifted pattern of the input data we introduce the shifting data process which insert the duplicated $w-1$ rows shift the original column expression data to be shifted input data of the suffix tree.

We start our shifting data process with checking if shifting window size is greater than 1, $w > 1$. If it is then, for each string S in set of strings $\{S_1^{"}, ..., S_{|R|}^{"}\}$, we prepend the special symbols * to the front of S and delete the last input character out of the string and insert $*y$ to terminator symbol where y equal to time we do this process for this string S which was $w-1$ times for each S. We define S_{x*y} to be the duplicate of original string $S^{"}$ of row x with y times duplicate so $S_{x*y} = *\wedge y S_{x1}^{"} S_{x2}^{"} ... S_{x|c|-y-1}^{"} \$x*y$, where $S_{xi}^{"}$ is the expression of string $S_x^{"}$ at position i and $*\wedge y$ means '*' y times. Then we insert these data to our original input data. Therefore after this process, our rows of input data matrix will extend from $|R|$ to $|wR|$. We also define input data with original data and shifted data as set of string $\{S_1^s, ..., S_{|wR|}^s\}$.

C. Suffix Trees and SCCC-Biclusters

In this part of work, we state how the maximal SCCC-Biclusters of the input data matrix M_{ij} corresponding to the node in the generalized suffix tree T built from the set of strings $\{S_1^s, ..., S_{|wR|}^s\}$ which is taken from the input data matrix considering its shifted data within shifting window size.

1) Every internal node in suffix tree T corresponds to one

row-maximal, right-maximal SCCC-Bicluster in matrix M with at least two rows. Since an internal node v in T have the common substring length $P(v)$ for each of it leaf. Therefore, each internal node v defines a SCCC-Bicluster that has $P(v)$ columns and $L(v)$ rows. And every right-maximal, row-maximal SCCC-Bicluster with at least two rows correspond to internal node in T.

2) An internal node corresponds to a maximal SCCC-Bicluster if and only if there is no suffix link from any node with the same value of $L(v)$ pointing to it. Since if there is an incoming suffix link from an internal node u to node v with $L(u) = L(v)$ then bicluster corresponds to v is already included in bicluster corresponds to u, $v \subset u$. So v is not maximal SCCC-Bicluster.
3) An internal node in T corresponds to a left-maximal SCCC-Bicluster if and only if it satisfies the Definition 13. Since an internal node v can be maximal SCCC-Bicluster from the fact 2) or in the fact that $L(u) < L(v)$ which lead v and u to be separate biclusters.

With all above facts considered, it leads us to the theorem adapted from [19] which was defined as follows:

Theorem 1. Every maximal SCCC-Bicluster with at least two rows corresponds to an internal node in the generalized suffix tree T that satisfies Definition 11, and each of these internal nodes defines a maximal SCCC-Bicluster with at least two rows. We define these nodes as N.

D. Insignificant SCCC-Biclusters

With the shifted data included, the resulted generalized suffix tree is more complicated. This tree leads the original CCC-Biclustering algorithm to report insignificant biclusters in its result. We point out the insignificant biclusters as follows:

1) Biclusters starts with *. All the shifted data we duplicated will include in these biclusters which have no original data, therefore these biclusters is insignificant for works.
2) Biclusters which start with data symbols but only included shifted data. These biclusters are duplicate of the original biclusters but small than the original one.
3) Biclusters that have only the data from the same row. These bicluster only include the original input data and its own shifted data, therefore this group of biclusters also marked as insignificant. (cluster which contain only data from 1 row of original data matrix)
4) The last group of insignificant biclusters are biclusters which have internal nodes, M (other than N), and other leaf nodes, L, from N. Where L is only from the shifted data of the same string as the leaf node of original data in M. These are the combination cases of second and third cases in the same N. Since the internal nodes M already defines the maximal SCCC-Bicluster, therefore these nodes which define smaller biclusters and pattern are already considered to be insignificant in second and third cases, therefore these are also insignificant.

E. SCCC-Biclustering: A Linear Time Biclustering Algorithm for Finding and Report All Maximal SCCC-Biclusters

Theorem 1 implies that there is an algorithm which can find and report all maximal SCCC-Biclusters of discretized and transformed gene expression matrix M in time linear and size of the input matrix (after inserted shifting data) since it corresponds to a suffix tree which has all these properties.

Algorithm 1 performed the alphabet transformation to the discrete input data as described in the part A then we do the shifting data process to extend the input data described in part B. After that our algorithm starts to build a generalized suffix tree from the set of strings $\{S_1^s,...,S_{|wR|}^s\}$ which obtained from the shifting input data process. We now check each internal node whether the conditions in Theorem 1 are met. Nodes that do not meet the required conditions are marked as "invalid." We marked the nodes started with * or only have the shifted data or the nodes from only the same string in this process. Then we marked biclusters which have internal node and other leaf nodes which satisfy 4) as "Invalid." Finally, we report our SCCC-Biclusters which corresponded to the valid internal nodes.

Algorithm1. SCCC-Biclustering

```
input: Discretized gene expression matrix M
1  Perform alphabet transformation and obtain
   $\{S_1^{''},...,S_{|R|}^{''}\}$.
2  Perform shifting data process and obtain
   $\{S_1^s,...,S_{|wR|}^s\}$.
3  Build a generalized suffix tree $T$ for $\{S_1^s,...,S_{|wR|}^s\}$
4  for each internal node $v \in T$ do
5      Mark node $v$ as "Valid."
6      Compute the string depth $P(v)$.
7  for each internal node $v \in T$ do
8      Mark node $v$ starts with * as "Invalid."
9      Compute the number of leaves $L(v)$ in the subtree
       rooted at $v$.
10     for each internal node $v \in T$ do
11         if there is a suffix link from $v$ to a node $u$ and
            $L(u) = L(v)$ then
12             Mark node $u$ as "Invalid."
13     for each internal node $v \in T$ do
14         Mark node $v$ with only shifted data or which
           leaf only come from the same string as "Invalid."
15     for each internal node $v \in T$ do
16         if node $v$ have internal nodes and leaf nodes as its
           child then
17             if leaf nodes are from shifted data and there is
               no leaf nodes in the internal node from other
               original data than the one of shifted data then
18                 Mark node $v$ as "Invalid."
19     for each internal node $v \in T$ do
20         if $v$ marked as "Valid" then
21             Report the SCCC-Bicluster corresponding to $v$.
```

F. Complexity Analysis of SCCC-Biclustering

In our algorithm, generalized suffix tree was used as our base data structure. With the proper implementation, generalized suffix tree was guaranteed to construct the suffix tree in linear time on the size of input matrix.

Our SCCC-Biclustering algorithm is corresponding to this suffix tree and performed by using the depth-first searches on this suffix tree. Since every tree structure has more internal nodes than leaf nodes, hence the running time of Algorithm1 is also linear time.

Since our algorithm extends the rows of input data in shifting process by shifting window size, w, times and with the complexity of the suffix tree, result in the total complexity of our algorithm to be in $O(w|R||C|)$ time. Also our algorithm uses size of input data matrix multiplied by w.

V. Conclusions

Our work opens a new path to find the biclusters which may be hidden in the expression data matrix. These hidden biclusters are due to some process in collection the gene expression data which leads to late activation of some expression in time. It may also be the type of genes which only activate some expression after the other related expression activated. This inspires the new idea within the bioinformatics path as well as some related others.

References

[1] G. J. McLachlan, K. Do, and C. Ambroise, “Analysing microarray gene expression data,” *Wiley Series in Probaility and Statistics*, 2004.

[2] Y. Cheng and G. M. Church, “Biclustering of expression data,” in *Proc. Eighth Int’l Conf. Intelligent Systems for Molecular Biology*, 2000, pp. 93-103.

[3] S. C. Madeira and A. L. Oliveira, “Biclustering algorithms for biological data analsis: a survey,” *IEEE/ACM Trans. Computational Biology and Bioinformatics*, vol. 1, no. 1, pp. 24-45, Jan. /Mar. 2004.

[4] I. V. Mechelen, H. H. Bock, and P. De Boeck, “Two-mode clustering methods: a structured overview,” *Statistical Methods in Medical Research*, vol. 13, no. 5, pp. 979-981, 2004.

[5] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, “Discovering local structure in gene expression data: the order-preserving submatrix problem,” in *Proc. Sixth Int’l Conf. Computational Biology*, 2002, pp. 49-57.

[6] L. ji and K. Tan, “Identifying time-lagged gene clusters using gene expression data,” *Bioinformatics*, vol. 21, no. 4, pp. 509-516, 2005.

[7] M. Koyuturk, W. Szpankowski, and A. Grama, “Biclustering gene-feature matrices for statistically significant dense patterns,” in *Proc. Eighth Int’l Conf. Research in Computational Molecular Biology*, 2004, pp. 480-484.

[8] J. Liu, W. Wang, and J. Yang, “Biclustering in gene expression data by tendency,” in *Proc. Third Int’l IEEE CS Computational Systems Bioinformatics Conf.*, 2004, pp. 182-193.

[9] J. Liu, W. Wang, and J. Yang, “A framework for ontology-driven subspace clustering,” in *Proc. ACM SIGKDD ’04*, 2004, pp. 623-628.

[10] J. Liu, W. Wang, and , J. Yang, “Gene ontology friendly biclustering of expression profiles,” in *Proc. Third IEEE CS Computational Systems Bioinformatics Conf.*, 2004, pp. 436-447.

[11] J. Liu, W. Wang, and J. Yang, “Mining sequential patterns from large data sets,” *Advances in Database Systems*, vol. 18, Kluwer Academic Publishers, 2005.

[12] S. Lonardi, W. Szpankowski, and Q. Yang, “Finding biclusters by random projections,” in *Proc. 15th Ann. Symp. Combinatorial Pattern Matching*, 2004, pp. 102-116.

[13] S. C. Madeira and A. L. Oliveira, “A linear time algorithm for biclustering time series expression data,” in *Proc. Fifth Workshop Algorithms in Bioinformatics*, 2005, pp. 39-52.

[14] T. M. Murali and S. Kasif, “Extracting conserved gene expression motifs from gene expression data,” in *Proc. Eighth Pacific Symp. Biocomputing*, 2003, vol. 8, pp. 77-88.

[15] A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler, “A systematic comparison and evaluation of biclustering methods for gene expression data,” *Bioinformatics*, vol. 22, no. 10, pp. 1282-1283, 2006.

[16] Q. Sheng, Y. Moreau, and B. De Moor, “Biclustering microarray data by gibbs sampling,” *Bioinformatics*, vol. 19, no. 2, pp. 196-205, 2003.

[17] A. Tanay, R. Sharan, and R. Shamir, “Discovering statiscally significant biclusters in gene expression data,” *Bioinformatics*, vol. 18, no. 1, pp. 136-144, 2002.

[18] C. Wu, Y. Fu, T. M. Murali, and S. Kasif, “Gene expression module discovery using gibbs sampling,” *Genome Informatics*, vol. 15, no. 1, pp. 239-248, 2004.

[19] S. C. Madeira, M. C. Teixeira, L. Sá-Correia, and A. L. Oliveira, “Identification of regulatory modules in time series gene expression data using a linear time biclustering algorithm,” *Computational Biology and Bioinformatics*, IEEE/ACM Transaction, vol. 7, no. 1, pp. 153-165, 2010.

[20] G. Getz, E. Levine, and E. Domany, “Coupled two-way clustering analysis of gene microarray data,” in *Proc. Natural Academy of Sciences Us*, 2000, pp. 12079-12084.

[21] S. C. Madeira, J. P. Gonçalves, A. L. Oliveira, “Efficient biclustering algorithms for identifying transacriptional regulation relationships using time series gene expression data,” *INESC_ID Tec. Rep.*, vol. 22, 2007.

[22] R. Peeters, “The maximum edge biclique problem is np-complete,” *Discrete Applied Math.*, vol. 131, no. 3, pp. 651-654, 2003.

Tustanah Phukhachee received the B.Eng. degree in computer engineering from King Mongut’s University of Technology North Bangkok in 2011. He is currently a graduate student at at King Mongkut's University of Technology Thonburi, Thailand. His research interests include computer algorithm, bioinformatics, data mining, and clustering.

Songrit Maneewongvatana received his Ph.D. in Computer Science from University of Maryland, College Park in 2001. He is currently an associate professor at King Mongkut's University of Technology Thonburi, Thailand. His research interests include clustering, data mining and learning technologies.