Proceedings of the 26th Annual International Conference of the IEEE EMBS
San Francisco, CA, USA • September 1-5, 2004

An Application of Zero-suppressed Binary Decision Diagrams to Clustering Analysis of DNA Microarray Data

Sungroh Yoon and Giovanni De Micheli
Computer Systems Laboratory, Stanford University, CA, USA

Abstract—**Clustering has been one of the most popular techniques to analyze gene expression data. The biclustering method is two-dimensional clustering of genes and experimental conditions to identify a group of genes that display a coherent behavior in some conditions. Although this method may provide additional insight overlooked by traditional clustering techniques, it is often computationally expensive to perform biclustering on practical gene expression data. In this work, we propose a novel biclustering technique that exploits the *zero-suppressed binary decision diagrams* (*ZBDDs*) to cope with such a computational challenge. The ZBDDs are a variant of the reduced ordered binary decision diagrams that have found a widespread use in optimization and verification of VLSI digital circuits. Our experimental results demonstrate that the ZBDDs can indeed extend the scalability of our biclustering algorithm substantially, thus enabling us to apply it to a wider spectrum of gene expression data.**

Keywords—**Clustering, gene expression analysis, ZBDD**

I. Introduction

Clustering is an unsupervised learning technique that has become a basic tool for researchers in the field of gene expression analysis. Although there is mature statistical literature on clustering, DNA microarray data has sparked the development of multiple new methods [1]. In particular, the biclustering technique [2,3,4,5,6,7] is one of the most promising innovations in this area [1]. The term *biclustering* was originally used to describe simultaneous clustering of both row and column sets in a data matrix [2]. The δ-biclustering [2] was the first application of this technique to gene expression data analysis. The δ-biclustering approach also included the notion of subspace clustering [7] to define a *bicluster*, which corresponds to a subset of genes displaying a coherent behavior under a subset of conditions. The term biclustering thus refers to the technique of two-dimensional subspace clustering.

The biclustering method may give additional biological insight that has been overlooked by conventional clustering approaches. For example, biclusters are more compatible with our understanding of cellular processes than other "global" clustering methods. This is because we often expect subsets of genes to be co-regulated and co-expressed under certain conditions, but to behave almost independently under other conditions [3]. Additionally, biclusters can overlap with each other, unlike more traditional hierarchical clusters. The biclustering method may thus be useful in recognizing reusable genetic "modules" that are mixed and matched in order to create more complex genetic responses [1].

Given the general notion of a bicluster, its formal definition varies depending on the clustering strategies applied. Two definitions are particularly related to our work. The δ-biclustering approach [2] defined a bicluster as a submatrix (of the complete data matrix) having a *mean squared residue* (*MSR*) score less than a given threshold. The pClustering technique [7] modeled a bicluster using the *pScore* that can summarize similarity between every pair of genes and experiments in the bicluster. These two definitions are in fact closely related. The pClustering method models more homogeneous clusters [7], and thus typically finds biclusters with lower MSR scores, meaning more coherence [8].

The cluster search problem is in general NP-hard [9], and the biclustering problem is no exception [2]. Most biclustering approaches including δ-biclustering therefore employed a heuristic to find *some* biclusters. In contrast, the pClustering approach employed a combinatorial approach to discover *all* biclusters on a given data matrix. Consequently, the δ-biclustering algorithm is more efficient, although the biclusters found by the pClustering method are better in terms of MSR scores, the measure of coherence [8].

In this work, we propose a clustering technique to find more coherent biclusters than the δ-biclustering method [2], without compromising efficiency. Our technique can find all biclusters existing on a given data set, too. The novelty of our method lies in the use of the *zero-suppressed decision diagram* (*ZBDDs*) [10], which can symbolically represent massive data and implicitly manipulate them.

II. Preliminaries

A. Definition of Bicluster and Biclustering Problem

We assume the reader is familiar with DNA microarray technologies [1]. To model more coherent biclusters than δ-biclustering, we use the concept of *pScore* [7]. Then our definition of bicluster is equivalent to that of pCluster [7]. The pScore of 2×2 matrix $X = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is defined as

$$pScore\,(X) = |(a-b)-(c-d)| = |(a-c)-(b-d)|. \quad (1)$$

Let U_G be the set of genes (rows) and U_E be the set of experiments (columns) in a gene expression data matrix $D \in \mathbf{R}^{U_G \times U_E}$. Suppose G and E are two sets such that $G \subseteq U_G$ and $E \subseteq U_E$. The pair $B = (G, E)$ denotes a submatrix of D, as shown in Fig. 1. The pair $B = (G, E)$ is called *bicluster* if we have $pScore\ (X) \le \delta$ for any 2×2 submatrix X in B and some $\delta \ge 0$.

This work was supported by a grant of Jerry Yang and Akiko Yamazaki.

An important property is that a subcluster of a pCluster is another pCluster for the same δ [7].

Given a parameter triplet (δ, M_G, M_E) representing a cluster threshold and a minimal number of genes and experiments, the objective is to find all pair $B = (G, E)$ such that (1) B is a bicluster with respect to δ; (2) B is not too small, namely $|G| \geq M_G$ and $|E| \geq M_E$; (3) B is *maximal* in the sense that its is not contained by others.

Unless otherwise stated, we use G and E to denote the gene and experiment set in a bicluster $B = (G, E)$, respectively, as shown in Fig. 1.

C. Pairwise Maximal Biclusters (PMBs)

The biclustering problem is in general intractable [2], but we can find *all* maximal biclusters in a $2 \times n$ or $n \times 2$ matrix in $O(nlogn)$ time [7]. Thus, we first find such biclusters, and then derive other biclusters from them. We call these special biclusters *pairwise maximal biclusters* (*PMBs*). The details on producing PMBs are beyond the scope of this paper, and we refer the interested to [2, 7].

As shown in Fig. 2, a *horizontal PMB* is a bicluster composed of two genes $\{g_i, g_j\}$ and a maximal (but not unique) set of experiments in which the two genes show a similar behavior. We refer to this maximal set as *horizontal seed* for genes $\{g_i, g_j\}$ or $E_m\{g_i, g_j\}$. For given $\{g_i, g_j\}$, there can be multiple $E_m\{g_i, g_j\}$ [7], and we denote by $\{E_m\{g_i, g_j\}\}$ the set of all those $E_m\{g_i, g_j\}$. By switching the roles of genes and experiments, *vertical PMB* and *vertical seed* are similarly defined.

D. Derivation of Biclusters from PMBs

To find a bicluster (G, E), we first derive E from horizontal PMBs and then calculate G from that E and vertical PMBs. This is due to the following observations in previous work:

1. E is a subset of a certain horizontal PMB [7, 8].
2. G is related to E and vertical PMBs through an analytical formula [8].

(It is typically less efficient to first get G and then calculate E from it, since $|U_G| >> |U_E|$ in typical gene expression data.)

We can discover any E by removing redundant elements from a certain horizontal seed. This is because for any E there exists a horizontal seed $E_m\{\cdot\}$ such that $E \subseteq E_m\{\cdot\}$ [7, 8]. We can efficiently perform this removal by an algorithm on the *trie*, a compact structure for strings [8].

To calculate G from E and vertical PMBs, we use the following formula [8]:

$$\{\text{all } G \text{ derivable from } E\} = \bigotimes_{\forall e_i, e_j \in E} \{G_m\{e_i, e_j\}\}, \qquad (2)$$

where ⊗ is a pairwise intersection operator on two sets of subsets $\underline{X}$ and $\underline{Y}$:[†]

$$\underline{X} \otimes \underline{Y} = \{X \cap Y \mid \forall X \in \underline{X} \text{ and } \forall Y \in \underline{Y}\}. \qquad (3)$$

[†] For example, {{0,1,2}, {2,3,4}} ⊗ {{0,2}, {4,5}} = {{0,2}, {2}, {4}}.

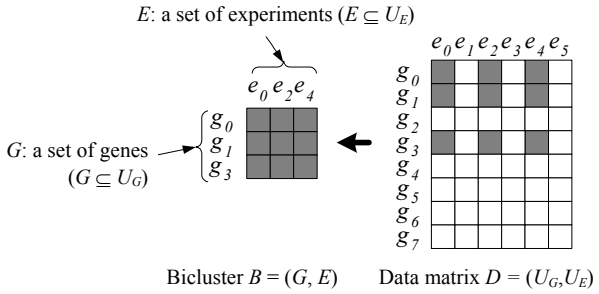


Fig. 1. The definition of bicluster.

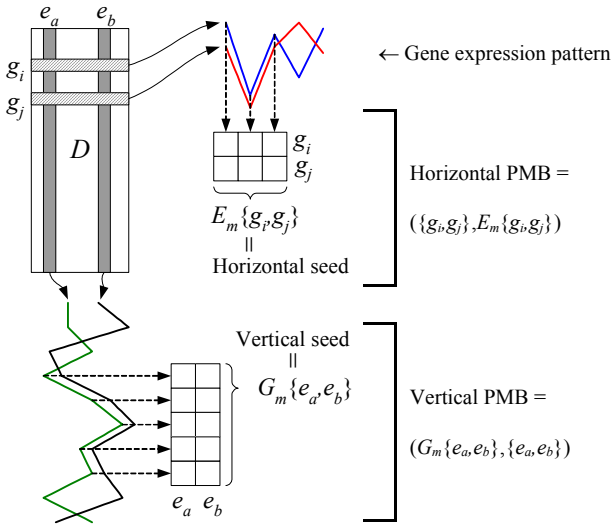


Fig. 2. Horizontal and vertical PMBs and seeds.

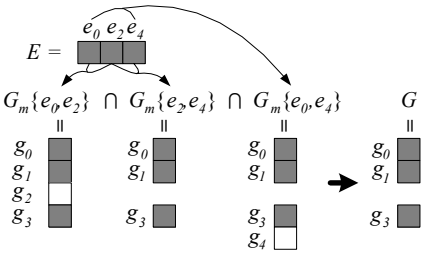


Fig. 3. The procedure to calculate G from E.

For example, consider the bicluster in Fig 1. Assuming each vertical seed $G_m\{\cdot\}$ is unique for simplicity, Fig. 3 depicts the procedure to calculate G from E.

The worst-case complexity of an algorithm to evaluate (2) is exponential in the number of columns in a data matrix. We therefore propose a new approach to better handle this computational challenge in the next section.

III. Methodology

A. ZBDD-based Representation of PMBs

We represent the vertical and horizontal PMBs by the *zero-suppressed binary decision diagrams (ZBDDs)* [10,11]. When ZBDDs are used, complexity of a problem no longer

depends on the size of the problem itself, but on that of its ZBDD representations, which often have mild growth with the problem size. Thus, ZBDDs may be used to efficiently solve many practical instances of intractable problems [11].

In many combinatorial problems, we need to manipulate sets of combinations [12]. Let $\mathbf{B} = \{0,1\}$. A *combination* of n elements is an n-bit vector $(x_1,\ldots,x_n) \in \mathbf{B}^n$, where the i-th bit reports whether or not the i-th element is contained in the combination. Thus, a set of combinations can be represented by a Boolean function f: $\mathbf{B}^n \rightarrow \mathbf{B}$. A combination given by the input vector $(x_1,\ldots,x_n)$ is contained in the set if and only if $f(x_1,\ldots,x_n) = 1$. In most combinatorial applications, the set of combinations are *sparse* in the following aspects:

- The sets contain only a small fraction of the 2^n possible bit vectors, and
- Each bit vector in the sets has many zeroes.

By exploiting both aspects, the ZBDDs provide a representation that is very efficient for representing and manipulating large-scale sets of combinations [10]. The reader can refer to [10, 11] for a more extensive treatment of ZBDDs.

Assuming the set of all genes is U_G, each $G_m\{\cdot\}$ corresponds to a combination of $|U_G|$ elements, and can be converted to a $|U_G|$-bit vector. For instance, $G_m\{e_1,e_3\} = \{g_0,g_2,g_4\}$ becomes (10101) assuming $U_G = \{g_0,g_1,g_2,g_3,g_4\}$. There can be multiple $G_m\{e_1,e_3\}$, and suppose we have another $G_m\{e_1,e_3\} = \{g_3,g_4\}$ which corresponds to (00011). The set of combinations {(10101), (00011)} can then be represented by the ZBDDs drawn in Fig. 4(a). The common elements between two sets are shared in the ZBDDs, as shown Fig. 4(b).

B. Efficient Implementation of ⊗ Operator through ZBDDs

We can implement the ⊗ operator with ZBDDs using basic set operators such as ∩ or ∪. They are recursively defined on ZBDDs with the trivial terminal cases such as $X \cap \varnothing = \varnothing$ or $Y \cup \varnothing = Y$ [10, 11].

We first show how to decompose a set of subsets into two smaller sets. Let $\underline{A}$ denote a set of subsets. We decompose $\underline{A}$ into $\underline{A}_1$ and $\underline{A}_0$, such that $\underline{A}_1$ has only those subsets containing a certain variable x, and $\underline{A}_0$ includes all the other subsets. In the ZBDD context, we can do this decomposition by recognizing two subgraphs of the topmost vertex: assuming the vertex represents x, the subgraph connected by the solid (dotted) line corresponds to $\underline{A}_1$ ($\underline{A}_0$). For example, in Fig. 4(a), $\underline{A}_1 = \{\{g_0,g_2,g_4\}\}$ and $\underline{A}_0 = \{\{g_3,g_4\}\}$, assuming $\underline{A} = \{\{g_0,g_2,g_4\}, \{g_3,g_4\}\}$ and $x = g_0$.

Based on this decomposition, we can recursively perform many operations on ZBDDs. For example, $\underline{A} \cup \underline{B} = (\underline{A}_0 \cup \underline{A}_1) \cup (\underline{B}_0 \cup \underline{B}_1) = (\underline{A}_0 \cup \underline{B}_0) \cup (\underline{A}_1 \cup \underline{B}_1)$, as shown in Fig. 5(a)[‡]. The problem of $\underline{A} \cup \underline{B}$ becomes two smaller problems, i.e., $(\underline{A}_0 \cup \underline{B}_0)$ and $(\underline{A}_1 \cup \underline{B}_1)$. We continue this process until encountering the terminal cases. The partial results are returned from the bottom, and eventually the final answer is available at the topmost vertex.

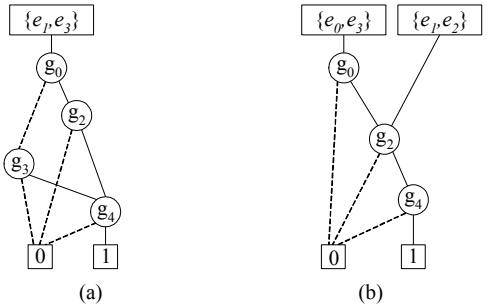


Fig. 4. ZBDD-based representations of PMBs.
The solid and dotted lines mean the 1-edge and 0-edge, respectively.

TABLE I
VERTICAL PMBS SHOWN IN FIGURE 4

Figure	Vertical PMBs ($G_m\{e_i, e_j\}$, $\{e_i, e_j\}$)
4 (a)	$(\{g_0, g_2, g_4\}, \{e_1, e_3\}), (\{g_3, g_4\}, \{e_1, e_3\})$
4 (b)	$(\{g_0, g_2, g_4\}, \{e_0, e_3\}), (\{g_2, g_4\}, \{e_1, e_2\})$

We can similarly compute $\underline{A} \otimes \underline{B}$ by recursively solving and merging four subproblems: $(\underline{A}_0 \otimes \underline{B}_0)$, $(\underline{A}_1 \otimes \underline{B}_0)$, $(\underline{A}_0 \otimes \underline{B}_1)$ and $(\underline{A}_1 \otimes \underline{B}_1)$. Fig. 5(b) depicts the decomposition for the topmost variable. The right subgraph includes $\underline{A}_1 \otimes \underline{B}_1$, and the left subgraph contains the others, since only $\underline{A}_1 \otimes \underline{B}_1$ can contain a subset with the topmost variable. For example, let $\underline{A}=\{\{g_0,g_2,g_4\},\{g_3,g_4\}\}$ and $\underline{B}=\{\{g_0,g_2,g_4\}\}$ from the examples in Fig. 4. Then $\underline{A} \otimes \underline{B}=\{\{g_0,g_2,g_4\},\{g_4\}\}$, as in Fig. 6.

III. EXPERIMENTAL RESULTS

A. Experiment Setup

The algorithms for the experiment are listed in TABLE II. We implemented the algorithms Z, P, and P+ with ANSI C++ on a 3.02 GHz Linux machine with 4 GB RAM. We used the CUDD [13] and EXTRA [14] packages for ZBDD

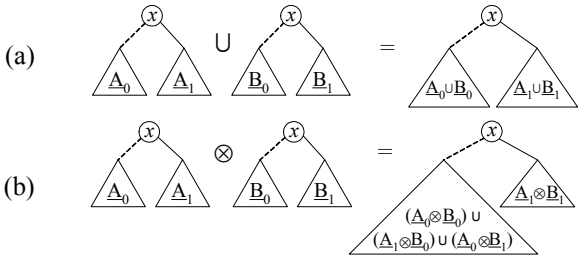


Fig. 5. Implementation of ⊗ using ZBDDs.
Replacing ∪ with ∩ in (a) gives the ZBDD representations for ∩.

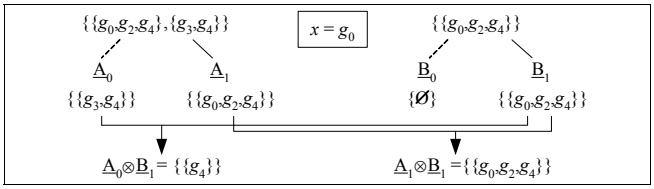


Fig. 6. An example of the ⊗ operation on ZBDDs.

[‡] For simplicity, we assume the topmost variables of two operands are the same in Fig. 5 (a) and (b).

libraries. We also downloaded an executable for the δ-biclustering algorithm from the website [2]. We tested the algorithms on the data sets shown in TABLE III. For creating the synthetic data set, we followed the procedure in [7]. The real expression data are from the yeast *Saccharomyces cerevisiae* cell cycle [15] and the human B-cell lymphoma [16], each of which was pre-processed by the method in [2].

B. Performance Evaluation via Synthetic Data Set

We executed the algorithms P and Z on D-synth to show the correctness and performance improvements of our algorithm. Fig. 7 shows the execution time. In Fig. 7(a), we varied the number of rows, fixing the number of columns to 30; in Fig. 7 (b), we instead changed the number of columns only, keeping the number of rows at 3,000. We observed that the algorithm Z substantially outperformed the algorithm P with respect to the execution time. We also verified that the algorithm Z could find all the biclusters predetermined and inserted into the data matrix.

C. Application to Real Gene Expression Data Sets

We applied the algorithms to D-yeast and D-lymph. Fig. 8(a) shows the time to discover the first 100 biclusters from D-yeast and D-lymph by the algorithm D and Z. Although the algorithm D is a heuristic and the algorithm Z is an exact method, their time efficiency was similar. The algorithm P could not respond in reasonable time in this experiment.

Fig. 8(b) is to show that the ZBDDs indeed extend the scalability of our algorithm. A higher value of δ means more number of PMBs and thus a larger problem size, since δ is a clustering threshold indicating a degree of screening [8]. For D-yeast used in this experiment, the appropriate value of δ was approximately 40-60, with $(M_G,M_E) \approx (30,5)$. Fig. 8(b) shows that our algorithm can handle the δ values in that range more efficiently than the algorithms P and P+, which were applicable only for smaller δ values.

IV. CONCLUSION

We proposed an efficient technique to find highly coherent biclusters on gene expression data. Our method was leveraged by the zero-suppressed binary decision diagrams, which can manage massive data efficiently. The experimental results established the effectiveness of our approach.

REFERENCES

[1] R. B. Altman and S. Raychaudhuri, "Whole-genome expression analysis: challenges beyond clustering," *Current Opinion in Structural Biology*, vol. 11, pp.340–347, 2001.

[2] Y. Cheng and G. M. Church, "Biclustering of expression data," in *Proc. ISMB'00*, pp. 93–103.

[3] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, "Discovering local structure in gene expression data: The order-preserving submatrix problem," in *Proc. RECOMB'02*.

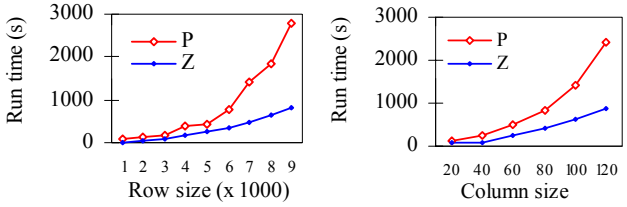
[4] G. Getz, E. Levine, and E. Domany, "Coupled two-way clustering analysis of gene microarray data," *Proc. Natl. Acad. Sci USA*, vol. 94, pp.12079–12084, 2000.

TABLE II
ALGORITHMS FOR EXPERIMENTS

ID	Description	Ref.
Z	Our ZBDD-based pClustering algorithm	-
P	The original pClustering algorithm	[7]
P+	The enhanced pClustering approach without ZBDDs	[8]
D	The δ-biclustering method	[2]

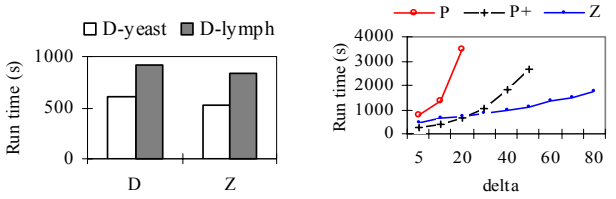
TABLE III
DATA SETS FOR EXPERIMENTS

ID	# rows	# columns	Origin	Ref.
D-synth	9,000	120	Synthetic	[7]
D-yeast	2,884	17	Yeast	[15]
D-lymph	4,026	96	Human B cells	[16]



(a) (b)

Fig. 7. The execution time comparison for the synthetic data.



(a) (b)

Fig. 8. The execution time comparison for the real expression data.

[5] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral bic lustering of microarray data: Coclustering genes and conditions," *Genome Research*, vol. 13, no. 4, pp.703–16, April 2003.

[6] A. Tanay, R. Sharan, and R. Shamir, "Discovering statistically significant biclusters in gene expression data," in *Proc. ISMB'02.*

[7] H. Wang, W. Wang, J. Yang, and P. S. Yu, "Clustering by pattern similarity in large data sets," in *Proc. ACM SIGMOD'02*, pp. 394–405.

[8] S. Yoon, C. Nardini, L. Benini, and G. De Micheli, "Enhanced pClustering and its applications to gene expression data," in *Proc. 4th IEEE Symposium on Bioinformatics and Bioengineering, BIBE '04*, in press.

[9] M. Sultan et al., "Binary tree-structured vector quantization approach to clustering and visualizing microarray data," *Bioinformatics*, vol. 18, pp. S111–S119, 2002.

[10] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in *Proc. IEEE/ACM DAC'93*, pp. 272–277.

[11] S. Minato, *Binary decision diagrams and applications for VLSI CAD*. New York, NY: Kluwer Academic Publishers, 1995.

[12] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*. Berlin, Germany: Springer, 1998.

[13] Available: http://vlsi.colorado.edu/~fabio/CUDD/

[14] Available: http://www.ee.pdx.edu/~alanmi/research/extra.htm

[15] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church, "Systematic determination of genetic network architecture," *Nature Genetics*, vol. 22, 281–285, 1999.

[16] A. Alizadeh et al., "Distinct types of diffuse large b-cell lymphoma identified by gene-expression profiling," *Nature*, vol. 4051, pp. 503–511, 2000.