A Pattern-Mining Method for High-Throughput Lab-on-a-Chip Data Analysis

Sungroh Yoon, Luca Benini, *Senior Member, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—Biochips are emerging as a useful tool for high-throughput acquisition of biological data and continue to grow in information quality and in discovering new applications. Recent advances include CMOS-based integrated biosensor arrays for deoxyribonucleic acid (DNA) expression analysis (Hassibi and Lee, 2005), (Schienle *et al.*, 2004), and active research is ongoing for the miniaturization and integration of protein microarrays (Kiyonaka *et al.*, 2004), (Rubina *et al.*, 2003), (Scrivener *et al.*, 2003), tissue microarrays (TMAs), (Chen *et al.*, 2004), (Shergill *et al.*, 2004), and fluorescence-based multiplexed cytokine immunoassays (Wang *et al.*, 2002). The main advantages of microfluidic lab-on-a-chip include ease of use, speed of analysis, low sample and reagent consumption, and high reproducibility due to standardization and automation. Without effective data-analysis methods, however, the merit of acquiring massive data through biochips will be marginal. The high-dimensional nature of such data requires novel techniques that can cope with the curse of dimensionality better than conventional data-analysis approaches. In this paper, the authors proposed a pattern-mining method to analyze large-scale biological data obtained from high-throughput biochip experiments. In particular, when a data set is given as a matrix, the method can find patterns appearing in the form of (possibly overlapping) submatrices of the input matrix. The method exploits the techniques developed for the symbolic manipulation of Boolean functions. Leveraged by this approach, the method can find, given a data matrix, all patterns that satisfy specific input parameters. The authors tested the method with several large-scale biochip data and observed that the proposed method outperforms the alternatives in terms of efficiency and the number of patterns discovered.**

***Index Terms*—Bioinformatics, biomedical signal analysis, biomedical transducers, computer-aided analysis, data management, logic design.**

I. Introduction

INTEREST in *in vivo* and *in vitro* applications of lab-on-a-chip, also called microfluidics-based biochips or biomicroelectromechanical system (bio-MEMS), is growing [15], [40]. The main advantages of this technology include ease-of-use, speed of analysis, low sample and reagent consumption, and high reproducibility due to standardization and automation. Biochips has become one of the standard tools for high-throughput acquisition of biological data, as is evident from the recent advances in integrated biosensor arrays [17], [35], protein microarrays [19], [33], [36], tissue microarrays (TMAs) [8], [37], and fluorescence-based multiplexed cytokine immunoassays [41].

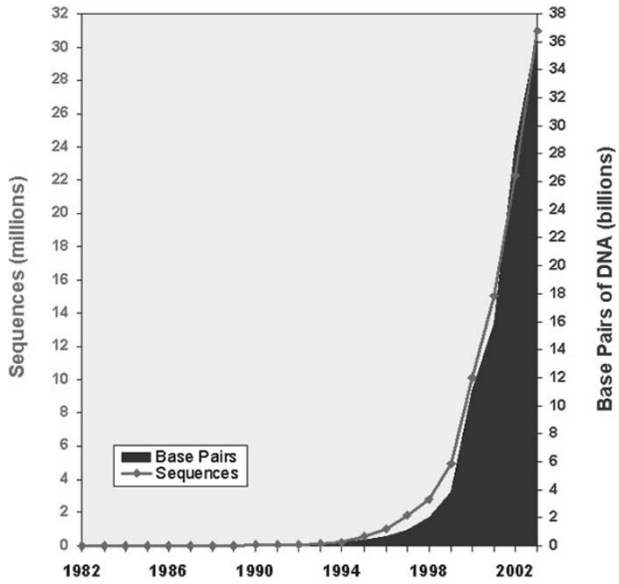


Fig. 1. Growth of GenBank database. The growth rate exceeds the pace set by Moore's Law.

However, the usefulness of this fascinating innovation may be limited without an effective means of analysis of the data obtained. In fact, technical breakthroughs in biotechnologies have already led to a rapid growth of biological data, both in size and complexity. For example, in recent years, the rate at which the GenBank database (http://www.ncbi.nlm.nih.gov/Genbank) has grown exceeds the pace set by Moore's Law, as seen in Fig. 1. Therefore, it is of utmost importance to have a fast and statistically robust data-analysis tool that can lead to breakthrough improvements in quality and time-to-market, by providing the designers of high-throughput biochips with the necessary feedback for the next design iteration in a timely manner.

Multiple new methods have been proposed to effectively analyze large-scale biological data obtained from high-throughput biotechnologies, despite the mature literature on traditional clinical-data analysis. This is partly because the data acquired from biochips often exhibit different characteristics from traditional clinical data. For instance, as seen in Fig. 2, the number of variables involved in a typical genomic study is far more than that of the observations, in contrast to a typical clinical study where there are normally more observations than variables

Manuscript received February 26, 2005; revised June 5, 2005. This work was supported in part by grants from J. Yang and A. Yamazaki. This paper was recommended by Associate Editor J. Zeng.

S. Yoon is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA (e-mail: sryoon@stanford.edu).

L. Benini is with the Department of Electrical Engineering and Computer Science (DEIS), University of Bologna, 40136 Bologna, Italy.

G. De Micheli is with the Integrated Systems Center, Ecole Polytechnique Fédérale de Lausanne (EPF Lausanne), CH-1015 Lausanne, Switzerland.

Digital Object Identifier 10.1109/TCAD.2005.855960

0278-0070/$20.00 © 2006 IEEE

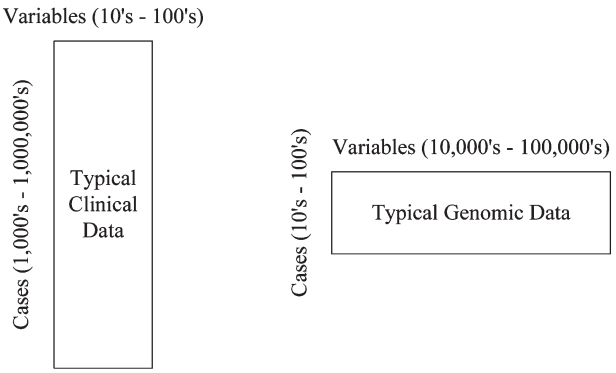


Fig. 2. Major difference between classic clinical studies and genomic studies [20]. In contrast to clinical data, genomic data often results in a highly underdetermined system.

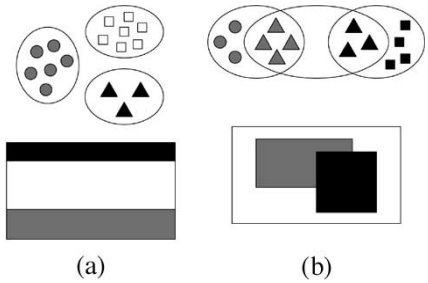


Fig. 3. Comparison between classic clusters and the patterns our method can find. (a) Objects are partitioned into mutually exclusive groups. (b) Patterns in our definition are allowed to belong to multiple groups.

[20]. Thus, in typical genomic studies, we often encounter the curse of dimensionality and the problem of identifying a highly underdetermined system.

Among the methods that have been proposed to handle this challenge, one of the most natural and in fact effective approaches is to focus only on subsets of the entire data [2]. By performing simultaneous clustering of rows and columns in a data matrix, we can discover some local structure appearing in the form of overlapping submatrices of the matrix. In this paper, we use the term pattern to refer to this local structure (see Fig. 3). In the literature, the local structure modeled by a submatrix is also termed coclusters, biclusters, or modules. The interested reader is directed to [23] for a review.

We observe that many patterns defined in the literature possess a common property. Suppose that $\mathfrak{D}$ is a certain condition under which a pattern, P, is defined. Here, we refer to the pattern P as homogeneous if any legitimate subpattern of P also satisfies the condition $\mathfrak{D}$. Examples of homogeneous patterns in the literature include conserved gene expression motifs (xMOTIFs) [29], δ-valid kj-patterns [7], gene expression module sampler (GEMS) [43], order-preserving submatrix problems (OPSMs) [3], order preserving (OP)-Clusters [21], and δ-pClusters [42], just to name a few.

Despite its relevance, the problem of homogeneous pattern mining is often computationally challenging. Let A be a matrix with row set R and column set C. The matrix A can be converted to a weighted bipartite graph $G = (V, E)$, where the vertex set $V = R \cup C$ and the edge set E consists of edge $\{i, j\}$ connecting row $i \in R$ and column $j \in C$ with weight a_{ij}. A submatrix of A then corresponds to a biclique in the graph G. To find not just any submatrix but a useful one, we need to consider individual elements of a submatrix, or equivalently, the edge weights of a biclique. Moreover, in order to avoid redundancy, we usually focus on finding maximal submatrices. Therefore, the problem of discovering patterns with certain semantics is at least as hard as that of finding the maximum edge biclique in a bipartite graph, a problem known to be NP-complete [23], [30].

In this paper, we propose a novel pattern-mining method that exploits the techniques commonly used for the symbolic manipulation of Boolean functions. The techniques have been reported to be useful in solving many practical instances of intractable problems [5], [6], [12], [25], [34]. In particular, we use the zero-suppressed binary decision diagrams (ZBDDs) [26], [27] to implicitly represent and manipulate massive intermediate data occurring in the pattern-mining process. Leveraged by this approach, our method can find, given a data matrix, all homogeneous patterns that satisfy specific input parameters. Especially, our method can find three types of homogeneous patterns, which are defined in such a way that they can serve as representative examples of the homogeneous patterns frequently encountered in the literature.

In our experiments, we first tested the proposed method with synthetic data sets to verify its validity. We then applied our method to some biological data in order to evaluate its applicability to actual biological data sets. We used gene-expression data obtained from genome chip experiments [13], [22]. This type of data is one of the most large-scale biochip data available. We observed that our method outperforms the alternative methods that are designed to find the same patterns, not only in terms of efficiency but also with respect to the total number of patterns discovered. In particular, we confirmed that the use of ZBDDs can greatly enhance the scalability of our approach and enable us to apply it to large-scale data sets.

The remainder of this paper is organized as follows. In Section II, we brief the reader on some biochip technologies in order to show the wide applicability of our method. Section III presents the formal definition of homogeneous patterns our method can find. In Sections IV and V, we explain at length the proposed method, which consists of essentially two stages. The first stage, which is detailed in Section IV, is to find special homogeneous patterns called atomic patterns. Section V presents the second stage of our method, which derives general (nonatomic) homogeneous patterns from the atomic patterns previously found. Section VI provides our experimental results, followed by the conclusion in Section VII.

II. Background: Data Acquisition by High-Throughput Biochips

After readout and preliminary data processing, biological data produced by high-throughput technologies are typically arranged in a matrix. Our method can analyze any type of biochip data, as long as the input data are represented as a matrix of real numbers. Here we present several examples, in order to provide an idea of the wide applicability of our approach. Some of these technologies have already been implemented

into a biochip, whereas others are currently under active research for miniaturization and integration.

One of the most well known and widely available is the deoxyribonucleic acid (DNA) microarray technology [13], [22], which enables us to monitor the expression levels of a large number of genes simultaneously, providing a global view of gene-expression information of the organism under study [2], [20], [31]. Depending upon the specific technology used, a DNA microarray data matrix reflects either absolute-expression levels (e.g., Affymetrix GeneChips [22]) or relative expression ratios (e.g., complementary DNA (cDNA) microarrays [13]) of thousands of genes under hundreds of experimental conditions. Recently, CMOS-based integrated DNA microarrays have been reported [17], [35], and the scale of integration will continue to grow.

The TMA technique enables researchers to extract small cylinders of tissue from histological sections and arrange them in a matrix configuration on a recipient paraffin block such that hundreds can be analyzed simultaneously [8], [37]. TMA thus allows the rapid and cost-effective validation of novel markers in multiple pathological tissue specimens.

The protein microarray is a crucial biomaterial for the rapid and high-throughput assay of many biological events where proteins are involved. In contrast to the DNA microarray, it has not been sufficiently established because of protein instability under the conventional dry conditions [19]. However, protein microarrays will eventually reveal vast amounts of information essential to the understanding of gene functions and products.

Other examples include the fluorescence-based multiplexed cytokine immunoassays [41] and ligand chips [33]. In particular, using the cytokine chip, cytokine expression in breast-cancer cells were examined and the chemokines associated with human cervical cancers were successfully identified [41].

III. Definitions and Overview

Our method is a generalization of some homogeneous pattern-mining techniques in the literature [3], [7], [9], [21], [29], [43], [44]. Thus, within a unified framework, our approach can find various types of homogeneous patterns. In particular, we focus on finding three specific types of homogeneous patterns in this paper. Their formal definitions are provided in Section III-A. Some biological intuition behind these definitions is presented in Section III-B. The problem statement and an overview of our approach will follow in Sections III-C and III-D, respectively.

A. Definition of Homogeneous Patterns

Throughout the paper, we let A denote an input data matrix of real numbers with set of rows $R = \{1, 2, \ldots, n\}$ and set of columns $C = \{1, 2, \ldots, m\}$. That is, $A \in \mathbb{R}^{n \times m}$. We also denote the matrix A by pair (R, C). We first provide a formal definition of a homogeneous pattern.

Definition 1: Given $A = (R, C)$, an input matrix, and $\mathfrak{D}$, a certain condition defined on a matrix, let pair $P = (I, J)$ denote a submatrix of A, namely, $I \subseteq R$ and $J \subseteq C$. The submatrix P is called a pattern appearing in A under $\mathfrak{D}$, if P satisfies the condition $\mathfrak{D}$.

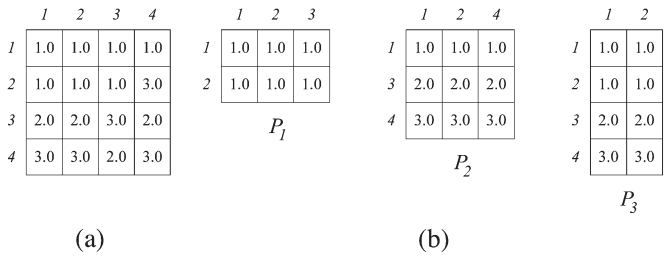


Fig. 4. Example of an input matrix and some patterns appearing in the matrix. The condition $\mathfrak{D}$ here defines a matrix in which the values on each row are constant. (a) Input matrix A. (b) Patterns P_1, P_2, P_3.

Example 1: Fig. 4(a) presents matrix $A \in \mathbb{R}^{4 \times 4}$ with $R = C = \{1, 2, 3, 4\}$. Let $\mathfrak{D}$ define a matrix in which the values on each row are constant. Fig. 4(b) shows P_1, P_2, P_3, some patterns appearing in the matrix A under the condition $\mathfrak{D}$.

Definition 2: Let P be a pattern appearing in matrix A under condition $\mathfrak{D}$. The pattern P is called homogeneous if any subset (or submatrix) of P is also a pattern appearing in the matrix A under the condition $\mathfrak{D}$.

Example 2: In Fig. 4(b), it can be easily verified that any submatrix of P_1, P_2, and P_3 is another pattern appearing in the matrix A under the same condition $\mathfrak{D}$, since the values on each row of such a submatrix remain constant. Thus, P_1, P_2, and P_3 are all homogeneous patterns.

We introduce the three types of homogeneous patterns which can be found by the pattern-mining method we are proposing in this paper. Table I is for a quick lookup of related information. In what follows, the term pattern always means a homogeneous pattern, unless otherwise stated.

1) Type-1 Patterns:

Definition 3: For any set S on $\mathbb{R}$, the range of S, denoted by $\text{RANGE}(S)$, is the difference between the largest and the smallest elements of S.

Definition 4: Given matrix $A = (R, C)$ and threshold $\tau \geq 0$, a type-1 pattern is a matrix denoted by (I, J), such that: 1) $I \subseteq R$ and $J \subseteq C$; and 2) for each $i \in I$, $\text{RANGE}(\{a_{ij} | \forall j \in j\}) \leq \tau$.

Example 3: Fig. 5 presents an input matrix and some type-1 patterns appearing in the matrix with respect to the parameter $\tau = 0.5$.

Type-1 patterns are a representative example of the patterns that have a one-row-based or one-column-based definition. Examples include a pattern with constant values on rows, as seen in Fig. 4, or with constant values on columns. In the literature, patterns such as xMOTIFs [29], δ-valid kj-patterns [7], and GEMS [43] belong to this type.

The reader can easily verify that any type-1 pattern is homogeneous. Furthermore, the following property holds for type-1 patterns.

Property 1: If (I_1, J_1) and (I_2, J_2) are both type-1 patterns with respect to τ, then the pattern $(I_1 \cup I_2, J_1 \cap J_2)$ is also type 1 with respect to τ.

Example 4: The patterns shown in Fig. 4(b) satisfies Definition 4 with respect to $\tau = 0$, and thus, P_1, P_2, and P_3 are all type-1 patterns. Let $P_1 = (I_1, J_1)$, $P_2 = (I_2, J_2)$, and $P_3 = (I_3, J_3)$. Then, $I_3 = I_1 \cup I_2$ and $J_3 = J_1 \cap J_2$. Thus, Property 1 holds for these patterns.

TABLE I
CLASSIFICATION OF HOMOGENEOUS PATTERNS

Pattern	Definition	Property	Example	Related patterns in the literature
Type 1	Definition 4	Property 1	Fig. 4, 5	xMOTIFs [29], δ-valid kj-pattern [7], GEMS [43]
Type 2	Definition 5	Property 2	Fig. 6	OPSM [3], OP-Cluster [21]
Type 3	Definition 6	–	Fig. 7	δ-bicluster [9], δ-pCluster [42, 46], FLOC cluster [44]

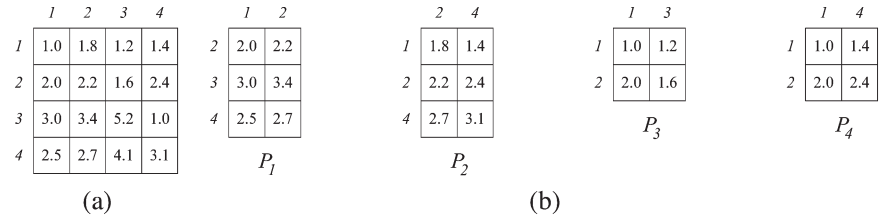


Fig. 5. Type-1 patterns appearing in the input matrix A (the parameter $\tau = 0.5$). (a) Input matrix A. (b) Type-1 patterns P_1, P_2, P_3, and P_4.

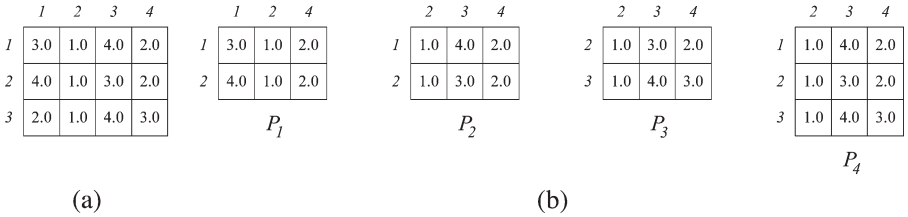


Fig. 6. Example of type-2 patterns. (a) Input matrix A. (b) Type-2 patterns P_1, P_2, P_3, and P_4.

2) Type-2 Patterns:

Definition 5: Given matrix $A = (R, C)$, let $J \subseteq C$ be a set of size $k \geq 2$ and let $\langle o_1, o_2, \ldots, o_k \rangle$ be a linear ordering of J. A type-2 pattern is a matrix denoted by (I, J) such that: 1) $I \subseteq R$; and 2) for each $i \in I$, $a_{io_1} > a_{io_2} > \cdots > a_{io_k}$.

Example 5: Fig. 6 presents a data matrix and type-2 patterns appearing in it. The order of the values on each row is preserved. For example, for $i \in I = \{1, 2\}$ in P_1, $a_{i1} > a_{i4} > a_{i2}$; for $i \in I = \{1, 2, 3\}$ in P_4, $a_{i3} > a_{i4} > a_{i2}$.

Type-2 patterns are a representative example of the patterns in which the order of the values (or some states defined by them) on a row or column is preserved for the other rows or columns as well. Examples in the literature include OPSMs [3] and OP-Clusters [21].

It can easily be verified that a type-2 pattern is homogeneous. In addition, the following property holds for type-2 patterns.

Property 2: If both (I_1, J_1) and (I_2, J_2) are type-2 patterns, then the pattern $(I_1 \cup I_2, J_1 \cap J_2)$ is also type 2.

Example 6: Property 2 holds for the patterns shown in Fig. 6(b). For example, let $P_2 = (I_2, J_2)$, $P_3 = (I_3, J_3)$, and $P_4 = (I_4, J_4)$. Then, it can be verified that $I_4 = I_2 \cup I_3$ and $J_4 = J_2 \cap J_3$.

3) Type-3 Patterns:

Definition 6: Given matrix $A = (R, C)$ and threshold $\tau \geq 0$, a type-3 pattern is a matrix denoted by $P = (I, J)$ such that: 1) $I \subseteq R$ and $J \subseteq C$; and 2) for any 2×2 submatrix $\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ in P, $|e - g - f + h| \leq \tau$.

Example 7: Fig. 7 shows a data matrix and some type-3 patterns appearing in the matrix with respect to the parameter $\tau = 1$.

Type-3 patterns are to model a matrix in which the elements exhibit some coherent behavior. Examples include a matrix in which the value of the elements fluctuate in harmony and a matrix in which all elements have the same value. Type-3 patterns in Definition 6 are in essence equivalent to δ-pClusters [42] and closely related to δ-biclusters[1] [9] and flexible overlapped biclustering (FLOC) clusters [44].

The reader can verify that type-3 patterns are homogeneous. However, Properties 1 and 2 do not necessarily hold for type-3 patterns. Also note that the same set of type-3 patterns can be found from input A and the transpose of A. This is because two matrices $\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ and $\begin{bmatrix} e & g \\ f & h \end{bmatrix}$ are indistinguishable in the definition since $|e - g - f + h| = |(e - g) - (f - h)| = |(e - f) - (g - h)|$.

B. Biology Behind the Definitions of Patterns

The three types of homogeneous patterns were defined in such a way that they can effectively capture important biological phenomena involved in various applications. For example, in gene-coregulation analysis, researchers are often interested in recognizing common fluctuations in the expression levels of multiple genes. Finding type-2 and type-3 patterns from gene-expression data matrices may be useful in this application. Discovering type-1 patterns can provide some biological insight

[1] δ-biclusters are not homogeneous patterns, since a subcluster of a δ-bicluster is not necessarily a δ-bicluster [9], [42]. However, δ-biclusters are included here because they also aim at modeling the coherent behavior of matrix elements, and it has been reported that δ-biclusters are closely related to δ-pClusters in many aspects [42], [46].

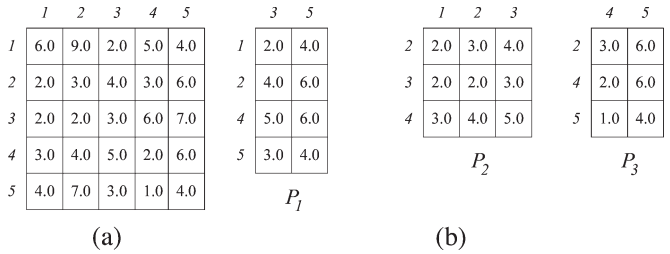


Fig. 7. Example of type-3 patterns (parameter $\tau = 1$). (a) Input matrix A. (b) Type-3 patterns P_1, P_2, and P_3.

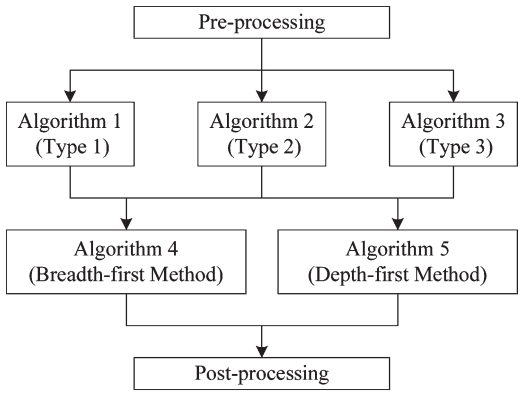


Fig. 8. Flowchart of our method. The first step (Algorithms 1, 2, and 3) is to find atomic patterns. The second step (Algorithms 4 and 5) is to derive nonatomic patterns.

for applications such as the task of marker-gene identification, where we are interested in correlating the activity of one or more genes to specific subphenotypes, and thus, finding genes expressed only in some phenotypes. For more examples, the reader can refer to [23], as well as the references listed in Table I.

C. Problem Statement

Given an input-data matrix $A = (R, C)$, a specific definition $\mathfrak{D} \in \{\text{Definition } 4, \text{Definition } 5, \text{Definition } 6\}$, and the parameters specified in $\mathfrak{D}$, the problem of pattern mining is to find all maximal homogeneous patterns $P = (I, J)$ appearing in A under $\mathfrak{D}$. We search only maximal[2] patterns or those that are not contained by other patterns as a submatrix, since nonmaximal patterns contain redundant information. Optionally, we can specify the minimum size of patterns in order not to generate patterns that are too small.

D. Overview of our Approach

Our pattern-mining algorithm consists of essentially two steps. The first step is to find special patterns called atomic patterns. The second step is to derive other general (nonatomic) patterns from the atomic patterns previously found. These two steps are detailed in Sections IV and V, respectively. Fig. 8 provides a flowchart of our method, and Tables II and III list related information for a quick reference.

[2]Formally, a pattern $P = (I, J)$ is called maximal if there is no pattern $P' = (I', J')$ such that $I \subseteq I'$ and $J \subseteq J'$ under the identical input conditions.

TABLE II
STEP 1—FINDING ATOMIC PATTERNS

Atomic pattern	Definition	Algorithm	Example
Type 1	Definition 7	Algorithm 1	Fig. 10
Type 2	Definition 8	Algorithm 2	Fig. 12
Type 3	Definition 9	Algorithm 3	Fig. 14

TABLE III
STEP 2—DERIVING NONATOMIC PATTERNS FROM ATOMIC PATTERNS

Method	Details	Algorithm	Based upon
Breadth-first	Section 5.3.2	Algorithm 4	Corollary 1
Depth-first	Section 5.3.3	Algorithm 5	Corollary 2

E. Notation

Table IV lists some important notations that will be used throughout the paper, especially in Section V.

IV. FINDING ATOMIC PATTERNS

Informally, an atomic pattern is represented by a matrix that has only one row (type 1) or two rows (types 2 and 3) but as many columns as possible. In this section, we provide the formal definition of atomic patterns and specific algorithms to find them.

A. Finding Type-1 Atomic Patterns

Definition 7: Given input matrix $A = (R, C)$ and threshold $\tau \geq 0$, a type-1 atomic pattern for row $i \in R$ is a one-row matrix, denoted by pair $P = (\{i\}, J)$, that satisfies the following: 1) P is a type-1 pattern on A; and 2) there is no J' such that $J' \supset J$ and $(\{i\}, J')$ is also a type-1 pattern.

The condition (2) in the above definition is not to generate those atomic patterns that are contained by others, since such patterns are redundant.

Algorithm 1 (Fig. 9) details our approach to find type-1 atomic patterns of Definition 4. The key idea of this algorithm is simple: when the elements of a set S are sorted and arranged in the corresponding order, range(S) is simply the absolute difference between the first and the last elements of S. The worst case complexity of the algorithm is polynomial in $|C|$, and the maximum number of atomic patterns found per row by Algorithm 1 is $(|C| - 1)$.

In **Lines 1–4**, the column indices are sorted in ascending order according to the value of the corresponding elements. The variables begin and end in **Lines 5–6** are to point to the first and the last elements of the subarray under consideration at some point. Inside the while loop in **Lines 7–16**, J, the column set of an atomic pattern, is generated as the variables begin and end are incremented. Note that multiple J can exist per row and overlap with each other. Since the array D is sorted, the algorithm only needs to compare in **Line 8** the first element ($D[\text{begin}]$) and the last element ($D[\text{end}]$), in order to see if all the elements in the subarray are similar. **In Lines 8–9**, the variable end is extended as long as $D[\text{end}].\text{val} - D[\text{begin}].\text{val} \leq \tau$. The algorithm reports J in **Line 11** or **Line 13**. **Lines 14–16** are to adjust the variable begin appropriately after one instance of

TABLE IV
NOTATIONS

Notation	Meaning
$A = (R, C)$	Input matrix A with row set R and column set C; $A \in \mathbb{R}^{\lvert R\rvert \times \lvert C\rvert}$.
$P = (I, J)$	Pattern P with row set I and column set J; $I \subseteq R$ and $J \subseteq C$.
$\mathcal{J}$	Set of column index sets.
v	Vertex in the lattice graph (Algorithms 4 and 5).
$v.I$	Row index set associated with vertex v.
$v.\mathcal{J}$	Set of column index sets associated with vertex v.
$\mathfrak{J}$	Function $\mathfrak{J}$ (Definition 11).
$\mathfrak{J}(I)$	Image of set I under function $\mathfrak{J}$; essentially, set of column index sets.
$\mathfrak{J}_1, \mathfrak{J}_2, \mathfrak{J}_3$	Function $\mathfrak{J}$ with explicit type specification.

Algorithm 1: Finding Type 1 atomic patterns for one row

input : $A = (R, C)$, a data matrix
input : $i \in R$, a row index
input : τ, a threshold
output: $J \subseteq C$, $(\{i\}, J)$ is a Type 1 atomic pattern

```
1  foreach j ∈ C do
2      D[j].val := a_ij;
3      D[j].ind := j;
4  Sort array D in ascending order with respect to the field val;
5  begin := 1;
6  end := 2;
7  while (end ≤ |C|) do
8      if (D[end].val − D[begin].val ≤ τ) then
9          end := end + 1;
10         if (end > |C|) then
11             Report J = {D[begin].ind, ..., D[end − 1].ind};
12     else
13         Report J = {D[begin].ind, ..., D[end − 1].ind};
14         repeat
15             begin := begin + 1;
16         until (begin = end) or
           (D[end].val − D[begin].val ≤ τ);
```

Fig. 9. Algorithm 1.

J is found, because multiple overlapping instances of J can be found for each row.

Example 8: Fig. 10(b) presents the type-1 atomic patterns discovered by Algorithm 1 from the data matrix in Fig. 5(a), repeated here in Fig. 10(a) for convenience. The parameter used is $\tau = 0.5$.

	1	2	3	4
1	1.0	1.8	1.2	1.4
2	2.0	2.2	1.6	2.4
3	3.0	3.4	5.2	1.0
4	2.5	2.7	4.1	3.1

(a)

Row $i \in R$	Column set $J \in 2^C$
1	$\{1,3,4\}$, $\{2,4\}$
2	$\{1,2,4\}$, $\{1,3\}$
3	$\{1,2\}$
4	$\{1,2\}$, $\{2,4\}$

(b)

Fig. 10. (a) Input matrix A. (b) Type-1 atomic patterns found from A by Algorithm 1 ($\tau = 0.5$).

Algorithm 2: Finding Type 2 atomic patterns for pair of rows

input : $A = (R, C)$, a data matrix
input : $q, r \in R$, row indices $(q \neq r)$
input : $min_J \geq 0$, minimum column size
output: $J \subseteq C$, $(\{q, r\}, J)$ is an atomic Type 2 pattern

```
1  foreach j ∈ C do
2      X̂[j].val := a_qj;
3      Ŷ[j].val := a_rj;
4      X̂[j].ind := Ŷ[j].ind := j;
5  Sort arrays X̂ and Ŷ in descending order with respect to the
   field val;
6  Construct sequence X = ⟨x_1, x_2, ..., x_|C|⟩ such that
   x_i = X̂[i].ind;
7  Construct sequence Y = ⟨y_1, y_2, ..., y_|C|⟩ such that
   y_i = Ŷ[i].ind;
8  Find Z = {Z | Z is an MCS of X and Y, |J| >= min_J};
9  foreach Z ∈ Z do
10     Convert Z = ⟨z_1, z_2, ...⟩ to J = {z_1, z_2, ...};
11     Return J;
```

Fig. 11. Algorithm 2.

B. Finding Type-2 Atomic Patterns

Definition 8: Given input matrix $A = (R, C)$, a type-2 atomic pattern for rows $i, k \in R (i \neq k)$ is a two-row matrix, denoted by pair $P = (\{i, k\}, J)$, that satisfies the following: 1) P is a type-2 pattern on A; and 2) there is no J' such that $J' \supset J$ and $(\{i, k\}, J')$ is also a type-2 pattern.

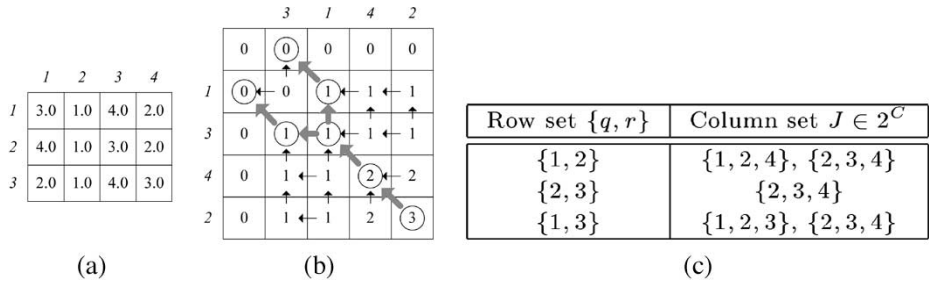
Our approach to find type-2 atomic patterns is outlined in Algorithm 2 (Fig. 11). The main problem is to find a largest two-row matrix in which the order of the values on each row is preserved. The key is to exploit algorithms to solve the problem of finding maximal common subsequences (MCSs) of two sequences [11], [16].

Besides an input-data matrix, Algorithm 2 takes an additional input parameter $\min_J$ to specify the minimum cardinality of the column set of an atomic pattern. This is to limit the total number of atomic patterns per pair of rows.

In **Lines 1–5**, the elements of each row are sorted with respect to their value, and the column indices are ordered accordingly. **Lines 6–7** are to convert arrays of column indices to sequences. In **Line 8**, an MCS-search algorithm is invoked. In **Lines 9–11**, each MCS found is converted to a set and returned.

The MCS problem has been extensively studied in the literature, and the typical solution relies on dynamic programming [11], [16]. The worst case complexity of an algorithm to solve the MCS problem is polynomial in the length of sequences [11], [16]. Some details of an MCS algorithm can be found in the following example.

Example 9: Fig. 12(c) presents the type-2 atomic patterns discovered by Algorithm 2 from the data matrix in Fig. 6(a), repeated here in Fig. 12(a) for convenience. The parameter used is $\min_J = 3$. We can solve the MCS problem by modeling it as



	1	2	3	4
1	3.0	1.0	4.0	2.0
2	4.0	1.0	3.0	2.0
3	2.0	1.0	4.0	3.0

(a)

	3	1	4	2
	0	(0)	0	0	0
1	(0)	0	(1)	1	1
3	0	(1)	(1)	1	1
4	0	1	1	(2)	2
2	0	1	1	2	(3)

(b)

Row set $\{q, r\}$	Column set $J \in 2^C$
$\{1, 2\}$	$\{1, 2, 4\}$, $\{2, 3, 4\}$
$\{2, 3\}$	$\{2, 3, 4\}$
$\{1, 3\}$	$\{1, 2, 3\}$, $\{2, 3, 4\}$

(c)

Fig. 12. (a) Input data. (b) Finding MCS for rows 1 and 2. (c) Type-2 atomic patterns found by Algorithm 2 ($\min_J = 3$).

a sequence-alignment problem [16]. In a sequence-alignment problem, the scores for a match, a mismatch, and a space should first be assigned. For the MCS problem, the scores for a match, a mismatch, and a space are one, zero, and zero, respectively [16]. Fig. 12(b) shows the dynamic-programming table for computing the MCS of two sequences $X = \langle 3, 1, 4, 2\rangle$ and $Y = \langle 1, 3, 4, 2\rangle$, derived from rows 1 and 2 of the input matrix A, respectively. We denote the entry in the ith row and the jth column by $D[i, j]$. We index the topmost row by $i = 0$ and use $j = 0$ to indicate the leftmost column. Let x_i and y_j denote the ith and the jth element of X and Y, respectively. Then, the optimal substructure of the MCS problem gives the following recursive formula [11]:

$$D[i,j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0, \\ D[i-1, j-1] + 1, & \text{if } i, j > 0 \text{ and } x_i = y_j. \\ \max(D[i-1, j], D[i, j-1]), & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

In addition, we place a traceback pointer ($\nwarrow, \uparrow, \longleftarrow$) in every entry $D[i, j]$ for $i > 0$ and $j > 0$, indicating where the value in the entry $D[i, j]$ originated (i.e., $D[i-1, j-1]$, $D[i-1, j]$, or $D[i, j-1]$). Each MCS corresponds to a traceback path from the largest element in the table, and this path is obtained by following the traceback pointers, which are indicated by the bold arrows in Fig. 12(b). In this particular example, two MCS exist, namely, $\langle 3, 4, 2\rangle$ and $\langle 1, 4, 2\rangle$. More details on this procedure can be found in [11] and [16].

One possible improvement of Algorithm 2 would be to consider "noisy ordering." That is, we can devise an algorithm that can rearrange elements with similar values in such a way that a longer MCS can emerge. This heuristic will help find atomic patterns with more columns, from which larger type-2 patterns can potentially be derived.

C. Finding Type-3 Atomic Patterns

Definition 9: Given input matrix $A = (R, C)$ and threshold $\tau \geq 0$, a type-3 atomic pattern for rows $i, k \in R (i \neq k)$ is a two-row matrix, denoted by pair $P = (\{i, k\}, J)$, that satisfies the following: 1) P is a type-3 pattern on A; and 2) there is no J', such that $J' \supset J$ and $(\{i, k\}, J')$ is also a type-3 pattern.

Algorithm 3 (Fig. 13) details our approach to find type-3 atomic patterns defined in Definition 6. This algorithm is equivalent to Algorithm 1, except for **Line 2**. An informal explanation is as follows. Algorithm 1 is to find a type-1 atomic pattern, or a one-row matrix in which the elements have similar values. Algorithm 3 is to find a two-row matrix in which any 2×2 submatrix $\begin{bmatrix} e & f \\ g & h \end{bmatrix}$ has similar values of $(e - g)$ and $(f - h)$, since $|e - g - f + h| = |(e - g) - (f - h)| \leq \tau$. Thus, we can use Algorithm 1 to find type-3 atomic patterns simply by subtracting the values in one row from the values in another and considering the result as a one-row matrix. This subtraction occurs in **Line 2** of Algorithm 3. Some details helpful to understand our informal proof can be found in [42].

Algorithm 3: Finding Type 3 atomic patterns for pair of rows

```
   input  : A = (R, C), a data matrix
   input  : q, r ∈ R, row indices (q ≠ r)
   input  : τ, a threshold
   output: J ⊆ C, ({q, r}, J) is an atomic Type 3 pattern
1  foreach j ∈ C do
2      D[j].val := a_qj − a_rj;
3      D[j].ind := j;
4  Sort array s in ascending order with respect to the field val;
5  begin := 1;
6  end := 2;
7  while (end ≤ |C|) do
8      if (D[end].val − D[begin].val ≤ τ) then
9          end := end + 1;
10         if (end > |C|) then
11             Report J = {D[begin].ind, ..., D[end − 1].ind};
12     else
13         Report J = {D[begin].ind, ..., D[end − 1].ind};
14         repeat
15             begin := begin + 1;
16         until (begin = end) or
           (D[end].val − D[begin].val ≤ τ);
```

Fig. 13. Algorithm 3.

Example 10: Fig. 14(b) presents the type-3 atomic patterns found by Algorithm 3 from the data in Fig. 7(a), repeated in Fig. 14(a). The parameter used is $\tau = 1$.

V. Our Pattern-Mining Algorithm

A. Overview

We can formulate the pattern-mining problem in terms of a binary relation.

Definition 10: Given $A = (R, C)$, an input-data matrix, and $\mathfrak{D}$, a specific definition of a pattern, $\mathfrak{R}_{\mathfrak{D}}$ is a binary relation on $2^R \times 2^C$

$$\mathfrak{R}_{\mathfrak{D}} = \{(I, J) | \text{ The pair } (I, J) \text{ forms a pattern appearing in } A \text{ under the definition } \mathfrak{D}\}. \quad (1)$$

Under this definition, the objective of pattern mining is to find the elements of the relation $\mathfrak{R}_{\mathfrak{D}}$. We aim at finding only maximal patterns, as stated in Section III-C.

Assume that we can find a function, denoted by $\mathfrak{J}$, that accepts as input $I \in 2^R$ and produces all maximal $J \in 2^C$ such that $(I, J) \in \mathfrak{R}_{\mathfrak{D}}$. Then, we may devise a naive algorithm that can provide all the elements of $\mathfrak{R}_{\mathfrak{D}}$: First, enumerate every $I \in 2^R$ and then feed it to the function $\mathfrak{J}$. Obviously, this approach is not feasible for a data matrix of nontrivial size since the powerset 2^R grows exponentially. Here, we explain how to improve this idea of exploiting the function $\mathfrak{J}$ so that we can apply it to mining homogeneous patterns appearing in large-scale data matrices. Formally, the definition of $\mathfrak{J}$ is as follows.

Definition 11: Given matrix $A = (R, C)$, $\mathfrak{J}$ is a function that maps $I \in 2^R$ to the image $\mathfrak{J}(I)$, where

$$\mathfrak{J}(I) = \left\{ J \in 2^C | (I, J) \in \mathfrak{R}_{\mathfrak{D}} \right. \\ \left. \text{and } \nexists J' \supset J \text{ s.t. } (I, J') \in \mathfrak{R}_{\mathfrak{D}} \right\}.$$

In Section V-B, we first explain how to define the function $\mathfrak{J}$ using the atomic patterns previously developed. In addition, we propose a novel technique to implement the function efficiently. The technique is based upon a data structure called ZBDDs [27]. Section V-C then presents how to exploit the function $\mathfrak{J}$ to find homogeneous patterns, avoiding the exhaustive enumeration of $I \in 2^R$. We propose two algorithms. One uses a breadth-first approach and the other employs a depth-first approach. Finally, Section V-D provides remarks on algorithm complexity and other issues.

B. Representation and Implementation of the Function $\mathfrak{J}$

We first introduce the operator $\otimes$, which is essentially the pairwise intersection of two sets of subsets but does not contain redundant subsets.

Definition 12: Let $\mathcal{T}$ and $\mathcal{U}$ be two sets of subsets. Also, let $\mathcal{Q} = \{T \cap U | \forall T \in \mathcal{T}, \forall U \in \mathcal{U}\}$. Then, the binary operator $\otimes$ on $\mathcal{T}$ and $\mathcal{U}$ is defined as follows:

$$\mathcal{T} \otimes \mathcal{U} = \mathcal{Q} - \{Q | \exists Q' \in \mathcal{Q} \text{ s.t. } Q' \supset Q\}. \tag{2}$$

Theorem 1: Let $\mathcal{T}, \mathcal{U}, \mathcal{W}$ be sets of sets. Then, $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} = \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$.

A proof of Theorem 1 is provided in Appendix B. The associative law thus holds for the operator $\otimes$, and it is trivial to show that the commutative law, $\mathcal{T} \otimes \mathcal{U} = \mathcal{U} \otimes \mathcal{T}$, holds. Consequently, we can develop the following notation.

Definition 13: The pairwise intersection of the k sets of sets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ is denoted by

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \cdots \otimes \mathcal{T}_k = \bigotimes_{i=1}^{k} \mathcal{T}_i. \tag{3}$$

In addition, we define the operator $\text{COVER}(S)$ for a set S in order to facilitate further explanation.

Definition 14: Given a set $S = \{s_1, s_2, \ldots, s_k\}$ with $k \geq 2$, $\text{COVER}(S)$ is a minimum edge cover of K_k, the complete graph with k vertices, in which the set of vertices corresponds to S.

Example 11: $\{\{0,1,2\}, \{2,3,4\}\} \otimes \{\{0,2\}, \{4,5\}\} = \{\{0,2\}, \{4\}\}$. Let $S_1 = \{1,2,3,4\}$ and $S_2 = \{10,11,12\}$. Then, a possible instance of $\text{COVER}(S_1) = \{\{1,3\}, \{2,4\}\}$, and an example of $\text{COVER}(S_2) = \{\{10,11\}, \{10,12\}\}$.

1) Redefining $\mathfrak{J}$ in Terms of Atomic Patterns: The image $\mathfrak{J}(I)$ defined in Definition 11 can be redefined using atomic patterns by the following theorem (see Appendix C for a proof).

Theorem 2: Let $\mathfrak{J}_1$, $\mathfrak{J}_2$, and $\mathfrak{J}_3$ denote the function $\mathfrak{J}$ for types 1, 2, and 3, respectively. Given input data $A = (R, C)$, the image of $I \in 2^R$, or $\mathfrak{J}(I)$, can be represented as follows.

1) When the set I has only one or two elements

$$\mathfrak{J}_1(\{r\}) = \{J | \text{ The pair } (\{r\}, J) \text{ is a type 1 atomic pattern for row } r \in R\} \tag{4}$$

$$\mathfrak{J}_2(\{q,r\}) = \{J | \text{ The pair } (\{q,r\}, J) \text{ is a type 2 atomic pattern for rows } q, r \in R\} \tag{5}$$

$$\mathfrak{J}_3(\{q,r\}) = \{J | \text{ The pair } (\{q,r\}, J) \text{ is a type 3 atomic pattern for rows } q, r \in R\}. \tag{6}$$

2) Otherwise

$$\mathfrak{J}_1(I) = \bigotimes_{\forall i \in I} \mathfrak{J}_1(\{i\}) \tag{7}$$

$$\mathfrak{J}_2(I) = \bigotimes_{\forall I' \in \text{COVER}(I)} \mathfrak{J}_2(I') \tag{8}$$

$$\mathfrak{J}_3(I) = \bigotimes_{\forall \{i,k\} \subseteq I} \mathfrak{J}_3(\{i,k\}). \tag{9}$$

To evaluate (7)–(9), we need to invoke the operator $\otimes$ at most $(|I| - 1)$, $(\lceil |I|/2 \rceil - 1)$, and $\binom{|I|}{2}$ times, respectively.

	1	2	3	4	5
1	6.0	9.0	2.0	5.0	4.0
2	2.0	3.0	4.0	3.0	6.0
3	2.0	2.0	3.0	6.0	7.0
4	3.0	4.0	5.0	2.0	6.0
5	4.0	7.0	3.0	1.0	4.0

(a)

Row set $\{q, r\}$	Column set $J \in 2^C$
$\{1,2\}$	$\{3,5\}$
$\{1,3\}$	$\{3,4\}$
$\{1,4\}$	$\{1,4\}, \{3,5\}$
$\{1,5\}$	$\{1,2\}, \{3,5\}$
$\{2,3\}$	$\{1,5\}, \{1,2,3\}$
$\{2,4\}$	$\{1,2,3,5\}, \{4,5\}$
$\{2,5\}$	$\{3,4,5\}$
$\{3,4\}$	$\{1,2,3\}$
$\{3,5\}$	$\emptyset$
$\{4,5\}$	$\{3,4,5\}$

(b)

Fig. 14. (a) Input matrix A. (b) Type-3 atomic patterns found from A by Algorithm 3 ($\tau = 1$).

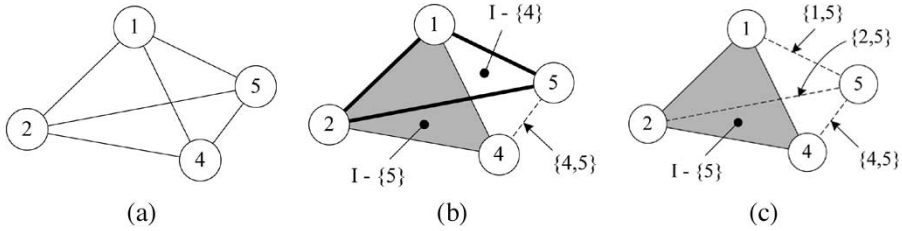


Fig. 15. Decomposition of the complete graph K_4. (a) Example 12: Theorem 2. (b) Example 13: Corollary 1. (c) Example 14: Corollary 2.

Example 12: To find the pattern P_1 in Fig. 7(b), we can use Theorem 2 and the atomic patterns presented in Fig. 14(b) as follows:

$$\begin{aligned}\mathfrak{J}_3(\{1,2,4,5\}) &= \mathfrak{J}_3(\{1,2\}) \otimes \mathfrak{J}_3(\{1,4\}) \otimes \mathfrak{J}_3(\{1,5\}) \\ &\quad \otimes \mathfrak{J}_3(\{2,4\}) \otimes \mathfrak{J}_3(\{2,5\}) \otimes \mathfrak{J}_3(\{4,5\}) \\ &= \{\{3,5\}\} \otimes \{\{1,4\},\{3,5\}\} \\ &\quad \otimes \{\{1,2\},\{3,5\}\} \otimes \{\{1,2,3,5\},\{4,5\}\} \\ &\quad \otimes \{\{3,4,5\}\} \otimes \{\{3,4,5\}\} \\ &= \{\{3,5\}\}.\end{aligned}$$

2) Enhancement by Dynamic Programming: We can reduce the number of the $\otimes$ operations required to evaluate the equations in Theorem 2 by storing and reusing intermediate results. This idea is similar to the concept of dynamic programming. In the equations in Theorem 2, we can see that the optimal substructure [11] appears, which is a hallmark of the applicability of dynamic programming.

For example, the process of realizing $\mathfrak{J}_3$ can be compared to that of decomposing a complete graph into its cliques. We start our explanation with revisiting Example 12. Let K_k denote the complete graph with k graph vertices. Suppose that we have the graph K_4, in which the vertices represent the elements of $I = \{1,2,4,5\}$ as shown in Fig. 15(a). In this figure, we can decompose the graph K_4 into $\binom{4}{2} = 6$ different K_2. This decomposition corresponds to evaluating (9). We, thus, evaluated $\mathfrak{J}_3(\{1,2,4,5\})$ using $\mathfrak{J}_3(\{1,2\}), \mathfrak{J}_3(\{1,4\}), \ldots, \mathfrak{J}_3(\{4,5\})$ in Example 12.

Alternatively, we can decompose K_4 into two different K_3 and one K_2 as shown in Fig. 15(b). The shaded triangle represents the set $I - \{5\} = \{1,2,4\}$ and the triangle indicated by bold lines represents $I - \{4\} = \{1,2,5\}$. This suggests a different way of evaluating $\mathfrak{J}_3(\{1,2,4,5\})$, namely, the evaluation using $\mathfrak{J}_3(\{1,2,4\})$, $\mathfrak{J}_3(\{1,2,5\})$, and $\mathfrak{J}_3(\{4,5\})$.

The alternative decomposition of $\mathfrak{J}_1$ and $\mathfrak{J}_2$ corresponding to Fig. 15(b) is simpler. Since $I = (I - \{4\}) \cup (I - \{5\})$, $\mathfrak{J}_t(I)$ is merely $\mathfrak{J}_t(I - \{4\}) \otimes \mathfrak{J}_t(I - \{5\})$, for each $t \in \{1,2\}$.

Corollary 1: Given input data $A = (R, C)$, let set $I \in 2^R$ and suppose that $i, k \in I$ and $i \neq k$. Then, the image $\mathfrak{J}_t(I)$ for each type $t \in \{1,2,3\}$ can be represented as follows:

$$\mathfrak{J}_1(I) = \mathfrak{J}_1(I - \{i\}) \otimes \mathfrak{J}_1(I - \{k\}) \tag{10}$$

$$\mathfrak{J}_2(I) = \mathfrak{J}_2(I - \{i\}) \otimes \mathfrak{J}_2(I - \{k\}) \tag{11}$$

$$\mathfrak{J}_3(I) = \mathfrak{J}_3(I - \{i\}) \otimes \mathfrak{J}_3(I - \{k\}) \otimes \mathfrak{J}_3(\{i,k\}). \tag{12}$$

When applying Corollary 1, we need to call the operator $\otimes$ only once (types 1 and 2) or twice (type 3), as long as the intermediate results $\mathfrak{J}(I - \{i\})$ and $\mathfrak{J}(I - \{k\})$ are available. In Section V-C2, we explain how to store and reuse intermediate results efficiently using a breadth-first search algorithm.

Example 13: We can apply Corollary 1 to the previous example as follows:

$$\begin{aligned}\mathfrak{J}_3(\{1,2,4,5\}) &= \mathfrak{J}_3(\{1,2,4\}) \otimes \mathfrak{J}_3(\{1,2,5\}) \\ &\quad \otimes \mathfrak{J}_3(\{4,5\}) \\ &= \{\{3,5\}\} \otimes \{\{3,5\}\} \otimes \{\{3,4,5\}\} \\ &= \{\{3,5\}\}.\end{aligned}$$

Fig. 15(c) shows another method of decomposing the graph K_4 for type-3 patterns. Here, K_4 is decomposed into one K_3 and three different K_2. The shaded triangle represents the set $I - \{5\} = \{1,2,4\}$ and the dotted lines the sets {1,5}, {2,5}, and $\{4,5\}$. This suggests a different way of evaluating $\mathfrak{J}_3(\{1,2,4,5\})$ using $\mathfrak{J}_3(\{1,2,4\})$, $\mathfrak{J}_3(\{1,5\})$, $\mathfrak{J}_3(\{2,5\})$, and $\mathfrak{J}_3(\{4,5\})$. The decomposition of $\mathfrak{J}_1$ and $\mathfrak{J}_2$ corresponding to Fig. 15(c) remains, in essence, the same as the previous case.

Corollary 2: Given input data $A = (R, C)$, let set $I \in 2^R$ and suppose that $k, l \in I$ and $k \neq l$. Then, the image $\mathfrak{J}_t(I)$ for each type $t \in \{1,2,3\}$ can be represented as follows:

$$\mathfrak{J}_1(I) = \mathfrak{J}_1(I - \{k\}) \otimes \mathfrak{J}_1(\{k\}) \tag{13}$$

$$\mathfrak{J}_2(I) = \mathfrak{J}_2(I - \{k\}) \otimes \mathfrak{J}_2(\{k,l\}) \tag{14}$$

$$\mathfrak{J}_3(I) = \mathfrak{J}_3(I - \{k\}) \otimes \left\{ \bigotimes_{\forall i \in I, i \neq k} \mathfrak{J}_3(\{i,k\}) \right\}. \tag{15}$$

In order to apply Corollary 2, we need to execute the operator $\otimes$ twice (types 1 and 2) or at most $(|I| - 1)$ times (type 3), as long as the result of $\mathfrak{J}(I - \{k\})$ is available. The number of $\otimes$ operations involved in the computation of $\mathfrak{J}_3$ in Corollary 2 is thus more than that in Corollary 1. However, it is easier to manage the partial results in Corollary 2, thus compensating for the larger number of $\otimes$ operations required. Section V-C3 presents a depth-first search algorithm, which exploits Corollary 2 to evaluate $\mathfrak{J}$ efficiently.

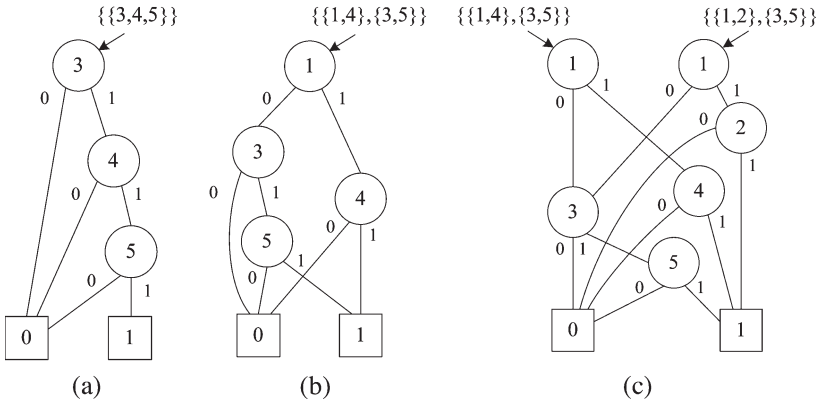


Fig. 16. ZBDD representation of atomic patterns: (a) $\mathfrak{J}_3(\{2,5\}) = \{\{3,4,5\}\}$. (b) $\mathfrak{J}_3(\{1,4\}) = \{\{1,4\},\{3,5\}\}$. (c) $\mathfrak{J}_3(\{1,4\}) = \{\{1,4\},\{3,5\}\}$ and $\mathfrak{J}_3(\{1,5\}) = \{\{1,2\},\{3,5\}\}$.

Example 14: We can apply Corollary 2 to Example 12 as follows:

$$\begin{aligned}\mathfrak{J}_3(\{1,2,4,5\}) &= \mathfrak{J}_3(\{1,2,4\}) \otimes \mathfrak{J}_3(\{1,5\}) \\ &\quad \otimes \mathfrak{J}_3(\{2,5\}) \otimes \mathfrak{J}_3(\{4,5\}) \\ &= \{\{3,5\}\} \otimes \{\{1,2\},\{3,5\}\} \\ &\quad \otimes \{\{3,4,5\}\} \otimes \{\{3,4,5\}\} \\ &= \{\{3,5\}\}.\end{aligned}$$

3) Efficient Implementation of the Operator $\otimes$ Using ZBDDs: We assume the reader to be familiar with the basic concepts of Boolean functions and with the data structures commonly used for the symbolic manipulation of such functions such as BDDs [5] and, in particular, ZBDDs [26]. Appendix A provides a brief introduction to ZBDDs. More extensive background material on this subject can be found in [12] and in [25]–[27].

In order to use ZBDDs to implement the operator $\otimes$, we first need to represent the operands of $\otimes$ by ZBDDs. A combination of m elements is an m-bit vector $\langle b_1, b_2, \ldots, b_m\rangle \in \mathbb{B}^m$, where $\mathbb{B} = \{0,1\}$. The ith bit reports whether the ith element is contained in the combination. Thus, a set of combinations corresponds to a Boolean function $\mathfrak{f} : \mathbb{B}^m \to \mathbb{B}$ and can be represented by ZBDDs. The operand of $\otimes$ is a set of column sets $\mathfrak{J}(I)$ and each column set $J \in \mathfrak{J}(I)$ can easily be converted to a combination as follows. Given input data $A = (R, C)$, assume $C = \{1, 2, \ldots, m\}$. Then, the set J corresponds to an m-bit vector $\langle b_1, b_2, \ldots, b_m\rangle$, where $b_i = 1$ if $i \in J$, and $b_i = 0$, otherwise. Representing this m-bit vector by ZBDDs is a standard procedure and is thus beyond the scope of this paper. We refer the interested reader to Appendix A and [25]–[27] for further details.

Example 15: In Fig. 14(b), $\mathfrak{J}_3(\{2,5\}) = \{\{3,4,5\}\}$. The set {3, 4, 5} can be converted to a 5-bit vector (00111) and represented by the ZBDD in Fig. 16(a). In the same example, $\mathfrak{J}_3(\{4,5\}) = \mathfrak{J}_3(\{2,5\})$. Thus, $\mathfrak{J}_3(\{4,5\})$ can be represented by the identical ZBDD for $\mathfrak{J}_3(\{2,5\})$ without creating a new one.

Example 16: In Fig. 14(b), $\mathfrak{J}_3(\{1,4\}) = \{\{1,4\},\{3,5\}\}$. This corresponds to the set of combinations {10010, 00101} and can be represented by the ZBDD in Fig. 16(b). Also, $\mathfrak{J}_3(\{1,5\}) = \{\{1,2\},\{3,5\}\}$ can share the part of the ZBDD for $\mathfrak{J}_3(\{1,4\})$, as shown in Fig. 16(c).

Next, we implement the operator $\otimes$ by directly manipulating the ZBDDs representing the operands. This allows us to avoid explicit enumeration of the intermediate results, thus providing a large speed-up over the conventional methods to represent and manipulate sets [26], [27]. As is often the case with the operators defined on ZBDDs, we define the operator $\otimes$ recursively. We first partition a set of combinations into two smaller sets of combinations. Let $\mathcal{T}$ be a set of combinations. We partition $\mathcal{T}$ into $\mathcal{T}_1$ and $\mathcal{T}_0$ with respect to the ith element b_i in such a way that $\mathcal{T}_1$ have all the combinations where $b_i = 1$, and $\mathcal{T}_0$ includes all the other combinations where $b_i = 0$. This partition can easily be done in a ZBDD by simply recognizing two subgraphs with respect to the topmost vertex. The subgraphs connected by the 1-edge and 0-edge correspond to $\mathcal{T}_1$ and $\mathcal{T}_0$, respectively. Based upon this partitioning, it follows that $\mathcal{T} \otimes \mathcal{U} = (\mathcal{T}_0 \otimes \mathcal{U}_0) \cup (\mathcal{T}_1 \otimes \mathcal{U}_0) \cup (\mathcal{T}_0 \otimes \mathcal{U}_1) \cup (\mathcal{T}_1 \otimes \mathcal{U}_1)$. Further implementation details can be found in [4], [5], [26], and [27].

C. Finding Homogeneous Patterns

We present two methods to find homogeneous patterns. Both methods utilize the function $\mathfrak{J}$ previously developed. Before providing the details of these methods in Sections V-C2 and V-C3, we present an example to explain the fundamental ideas common in both methods.

1) An Example of Finding Type-3 Patterns: Fig. 17 shows the process for finding the patterns in Fig. 7(b) from the data matrix in Fig. 7(a), in which the set of rows $R = \{1,2,3,4,5\}$. In the graphs shown in the figure, each vertex v has two associated fields, namely $v.I$ and $v.\mathcal{J}$. The field $v.I$ is to save a set of rows, and the field $v.\mathcal{J}$ is to store the image $\mathfrak{J}(v.I)$. The level of the vertex v is defined as the cardinality of $v.I$. Also, we connect vertex v_1 at level l and vertex v_2 at level $l+1$ by an edge if $v_1.I \subset v_2.I$.

Fig. 17(a) presents a graph in which each vertex represents an elements in 2^R and a vertex is connected to others by the above rule. For example, $v.I = \{1,2\}$ for the vertex v indicated by "12." This vertex is connected to the vertices indicated by "123," "124," and "125."

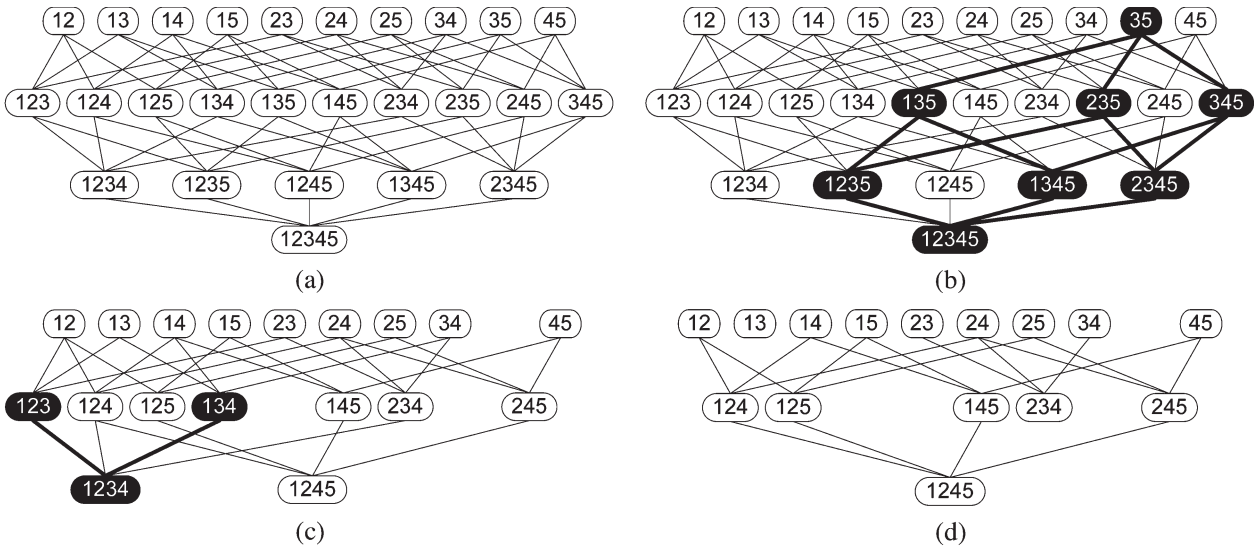


Fig. 17. Process of finding the patterns presented in Fig. 7(b). This is only for the explanation of the idea, and in practice, we do not need the graph in its entirety all the time. Refer to Algorithms 4 and 5 for details.

We can make two key observations in the graph constructed as above. First, not all vertices need to be examined. Thus, we can avoid exhaustive enumeration of $I \in 2^R$. Second, the intermediate results required to apply Corollaries 1 and 2 are available from the vertices at the previous level.

The first observation is based upon the following fact: If $\mathfrak{J}(I) = \emptyset$, then $\mathfrak{J}(I') = \emptyset$ for all $I' \supseteq I$. This is because if the pair (I, J) does not represent a homogeneous pattern, then the pair (I', J) with $I' \supseteq I$ cannot be a homogeneous pattern, either. For example, in Fig. 14(b), we know that $\mathfrak{J}_3(\{3,5\}) = \emptyset$. Thus, it is possible to conclude that $\mathfrak{J}_3(I) = \emptyset$ for all $I \supseteq \{3,5\}$. Consequently, we can eliminate any vertex v such that $v.I \supseteq \{3,5\}$. In the graph in Fig. 17(b), the vertices to be deleted are indicated.

The example in Fig. 17(c) shows that another vertex-elimination process is possible, starting from the vertices at level 3, namely, "123" and "134." The vertex "123" should be deleted because $\mathfrak{J}_3(\{1,2,3\}) = \mathfrak{J}_3(\{1,2\}) \otimes \mathfrak{J}_3(\{1,3\}) \otimes \mathfrak{J}_3(\{2,3\}) = \emptyset$. We can remove the vertex "134" similarly. Thus, any vertex v such that $v.I \supseteq \{1,2,3\}$ or $v.I \supseteq \{1,3,4\}$ can be deleted. Finally, the graph in Fig. 17(d) shows the vertices that remain undeleted. It is these vertices to which we apply the function $\mathfrak{J}$ to find homogeneous patterns. Since the remaining vertices correspond to all $I \in 2^R$ that can potentially be the row set of a homogeneous pattern, applying the function $\mathfrak{J}$ to these vertices enables us to find all the homogeneous patterns that satisfy the input parameters specified.

To exploit the intermediate results stored in the vertices at the previous level, the breadth-first algorithm in Section V-C2 starts with the vertices at level 2 and proceed to level $l+1$ from level l only after no vertex at level l is left. This is compatible with the decomposition of $\mathfrak{J}$ in Corollary 1. In contrast, the depth-first algorithm in Section V-C3 starts with vertex v at level 2 and proceeds until the algorithm examines all the vertices whose I set contains $v.I$. Then, the algorithm starts with another vertex at level 2. This algorithm fits with the decomposition of $\mathfrak{J}$ in Corollary 2. Both algorithms find the same homogeneous patterns, although one can be faster than the other, depending upon the specific input data matrix and parameters used.

One important comment is in order. Obviously, it is not realistic to construct the graph like the one in Fig. 17(a) in its entirety, especially when the set R has many elements. The examples in Fig. 17 are only for explanation. As will be described in Algorithms 4 and 5 (Figs. 18 and 19, respectively), the breadth-first and the depth-first algorithms do not need to examine all the vertices simultaneously.

2) Breadth-First Algorithm: Algorithm 4 details our breadth-first approach to find homogeneous patterns. The input is a data matrix, pattern type, and parameters for atomic-pattern generation. The output are homogeneous patterns found from the input data matrix.

In **Line 1**, atomic patterns are generated by the algorithms explained in Section IV with the input parameters.

In **Lines 2–11**, the base vertices at level 2 are generated. Each vertex v has three associated data fields. The fields $v.I$ and $v.\mathcal{J}$ are the same as explained in the previous section. The field v.level is to store the level of the vertex v. For type-2 or type-3 patterns, a new vertex is created for each pair of rows, unless no atomic pattern exists for the pair. The base vertices for type-1 patterns also start at level 2 by merging two atomic patterns.

In **Lines 13–34**, the algorithm iterates for each level and performs the following for each vertex at level l. In **Lines 16–17**, the algorithm reports any candidate patterns obtained from the previous iteration. In **Lines 18–34**, new vertices appearing at level $l+1$ are generated. To this end, the algorithm examines two vertices v_i and v_j at level l. **Lines 21–22** are to test if the two vertices are qualified to create a new vertex at level $l+1$. As long as the sets $v_i.I$ and $v_j.I$ have the same elements but one, the vertices v_i and v_j can create a new vertex at the next level. Since a vertex at level $l+1$ should have only one more row than a vertex at level l, if the union of $v_i.I$ and $v_j.I$ has more than $l+1$ elements, the two vertices v_i and v_j cannot spawn a new vertex in the next level. For example, if $v_i.I = \{1,2,3\}$ and $v_j.I = \{1,2,4\}$, then these two vertices

Algorithm 4: Breadth-first pattern mining algorithm

```
   input  : A = (R, C), a data matrix
   input  : type ∈ {1, 2, 3}, pattern type
   input  : parameters for atomic pattern generation
   output: homogeneous patterns
 1 Generate atomic patterns (see Section 4);
 2 foreach {q, r} ⊆ R do
 3     if type = 1 then
 4         J := ℑ1({q}) ⊗ ℑ1({r});
 5     else
 6         J := ℑ_type({q, r});
 7     if J ≠ ∅ then
 8         Create vertex v;
 9         v.I := {q, r};
10         v.J := J;
11         v.level := 2;
12 maxLevel := |R|;
13 for l = 2 to maxLevel do
14     for i = 1 to numVertices in level l do
15         v_i := i-th vertex in level l;
16         foreach J ∈ v_i.J do
17             Collect pattern (v_i.I, J);
18         if l < maxLevel then
19             for j = i + 1 to numVertices in level l do
20                 v_j := j-th vertex in level l;
21                 I := v_i.I ∪ v_j.I;
22                 if |I| ≠ l + 1 then next;
23                 if a vertex for I exists then next;
24                 if type = 3 then
25                     e_i := the element in I − v_i.I;
26                     e_j := the element in I − v_j.I;
27                     J := v_i.J ⊗ v_j.J ⊗ ℑ3({e_i, e_j});
28                 else
29                     J := v_i.J ⊗ v_j.J;
30                 if J ≠ ∅ then
31                     Create vertex v;
32                     v.I := I;
33                     v.J := J;
34                     v.level := l + 1;
35         Remove v_i;
36 Remove redundancy and return remaining patterns;
```

Fig. 18. Algorithm 4.

Algorithm 5: Depth-first Pattern Mining Algorithm

```
   input  : A = (R, C), a data matrix
   input  : type ∈ {1, 2, 3}, pattern type
   input  : parameters for atomic pattern generation
   output: homogeneous patterns
 1 Generate atomic patterns (see Section4);
 2 foreach {q, r} ⊆ R do
 3     Create vertex v;
 4     v.I := {q, r};
 5     ConstructTrieInPreorder(v);
 6     delete v;
 7 Remove redundancy and return remaining patterns;
 8 procedure ConstructTrieInPreorder (vertex v)
 9 begin
10     if |v.I| = 2 then
11         if type = 1 then
12             v.J := ℑ1({q}) ⊗ ℑ1({r});
13         else
14             v.J := ℑ_type(v.I);
15     else
16         vertex p := v.parent;
17         k := the element in v.I − p.I;
18         if type = 1 then
19             v.J := p.J ⊗ ℑ1({k});
20         else if type = 2 then
21             k' := any element in p.I;
22             v.J := p.J ⊗ ℑ2({k, k'});
23         else
24             v.J := p.J ⊗ (⊗_{∀i∈p.I} ℑ3({i, k}));
25     if v.J = ∅ then return;
26     foreach J in v.J do
27         Collect pattern (v.I, J);
28     l := the "largest" element in v.I wrt a total order ≺;
29     I := {i | i ∈ R and l ≺ i};
30     foreach i ∈ I do
31         create vertex w;
32         w.I := v.I ∪ {i};
33         w.parent := v;
34         ConstructTrieInPreorder(w);
35         delete w;
36 end
```

Fig. 19. Algorithm 5.

can create a new vertex v at level 4 with $v.I = \{1, 2, 3, 4\}$. In contrast, if $v_i.I = \{1, 2, 3\}$ and $v_j.I = \{1, 4, 5\}$, then they cannot generate a new vertex at level 4, because the row sets differ by two elements. This way of creating new vertices is to avoid exhaustive enumeration. In **Line 23**, if the two vertices v_i and v_j are eligible for creating a new vertex, the algorithm sees whether the corresponding vertex already exists or not. If not, the algorithm computes the set $\mathcal{J}$ for this new vertex by Corollary 1 in **Lines 24–29**. In **Lines 30–34**, the new vertex v is actually created and stored for further reference in the next iteration, if the set $\mathcal{J}$ is not empty. Otherwise, no new vertex is created. This corresponds to removing all the values downstream of the vertex v in which $v.\mathcal{J} = \emptyset$ [e.g., Fig. 17(b) and (c)].

In **Line 35**, a vertex is deleted as soon as it becomes of no use. Thus, the algorithm can keep at most two levels of the vertices at a time, rather than the entire graph.

In **Line 36**, any redundant patterns are removed and the remaining patterns are returned.

3) Depth-First Algorithm: In Section V-C2, we explained our breadth-first approach. In this section, we introduce an alternative pattern-mining algorithm using a depth-first approach. We start the description with the examples in Figs. 17 and 20. In order to visit the vertices in the depth-first sense, we need to restructure the graph. In particular, we remove some edges from the graph in Fig. 17(a) so that the graph becomes the trie in Fig. 20(a). A trie [1] is a special structure for representing sets of words. Here, we regard the set $I \in 2^R$ as a word assuming a total order among the elements in R. For instance, we can assume the total order $1 \prec 2 \prec 3 \prec 4 \prec 5$ for the set $R = \{1, 2, 3, 4, 5\}$. Hence, the set $I = \{1, 2, 3\}$ corresponds to the word "123." This word is inserted into the trie as the descendant of the word "12" and as the parent of the words "1234" and "1235," as shown in Fig. 20(a).

Algorithm 4 provides the details of our depth-first approach. In **Lines 2–6**, the algorithm traverses the trie in preorder. More precisely, our algorithm constructs the trie in preorder rather than traverses it. In other words, the algorithm creates vertices

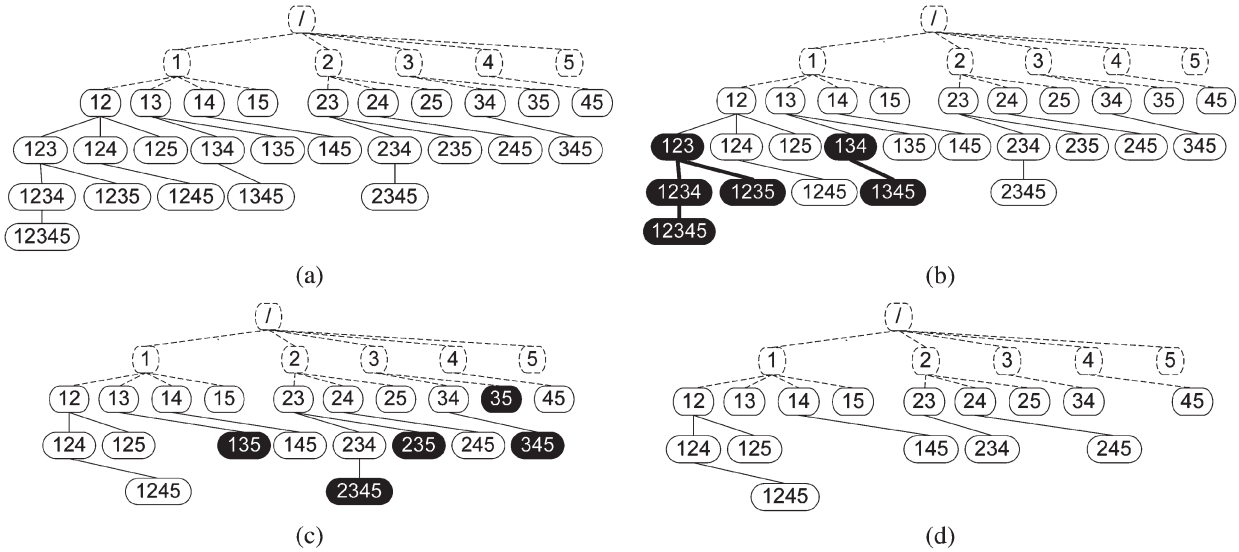


Fig. 20. Example for explaining the depth-first pattern-mining algorithm.

whenever necessary and deletes them afterwards rather than keeping the trie in its entirety all the time. In **Line 7**, the algorithm reports the homogeneous patterns produced after removing redundant patterns, if any.

For each vertex v encountered in this preorder construction of the trie, the algorithm performs the following (**Lines 8–36**). The algorithm computes $v.\mathcal{J}$ by Corollary 2 in **Line 18–24**. If the set $v.\mathcal{J}$ is empty, the algorithm does not proceed to examining the descendant vertices and returns to the parent vertex (**Line 25**). This is equivalent to deleting all the descendant vertices in Fig. 20(b). If the set $v.\mathcal{J}$ is not empty, the algorithm produces homogeneous patterns $(v.I, J)$ for all $J \in \mathfrak{J}(v.I)$ in **Lines 26–27**. Then, the algorithm creates a list of the descendant vertices in **Lines 28–29**. This step is necessary because the algorithm does not keep the entire trie all the time, and thus, the vertex v is not already connected to its children. The "largest" element in **Line 28** means the "largest" element in the total order we are assuming among the elements of R. For example, the largest element of the set $\{1, 2, 4\}$ is the element "4," assuming $1 \prec 2 \prec 4$. In **Lines 30–35**, the algorithm creates the descendant vertices and visits them to repeat the steps performed in **Lines 8–36**.

D. Remarks

The pattern-mining problem addressed in this paper is related to the problem of finding the maximum edge biclique in a bipartite graph, a problem known to be NP-complete [23], [30]. Although the worst case complexity of Algorithms 4 and 5 is exponential in the number of rows in the input data matrix, the execution time on typical benchmarks is practical, as will be shown in Section VI. This is due to efficient techniques, such as the ZBDD-based symbolic manipulations and the dynamic-programming approach, which enable us to avoid the exhaustive and explicit enumeration of the intermediate results. In particular, the role of the ZBDDs is crucial in this study. Without using the ZBDDs, it would not be possible to achieve the efficiency that the current implementation of our algorithm shows.

In fact, reduced ordered BDDs (ROBDDs) and variants such as ZBDDs have been widely used to solve many practical instances of intractable problems [12], [25], [27]. Some data-analysis methods recently proposed [28], [45] rely on this idea of managing massive data through the symbolic representation of Boolean functions. In particular, the method proposed in this study is a generalization and extension of our earlier work [45], which focused only on finding type-3 patterns.

Our pattern-mining algorithms discussed so far are exact in the sense that they can find all the patterns that satisfy specific input parameters. If desired, it is possible to employ a heuristic algorithm that runs quickly but can find only a subset of the possible patterns. For example, we can implement the "greedy" $\otimes$ operator that reports only k largest (in terms of cardinality) sets, which will make the cardinality of $\mathcal{J}$ decrease. We can also utilize a measure of overlap such as Jaccard's coefficient [24] to avoid generating "similar-looking" atomic patterns, thus reducing the number of atomic patterns considered in later steps.

VI. Experimental Results

We implemented our method in C++ on a 3.06-GHz Linux machine with 4-GB RAM. We used the libraries provided by the BDD packages Colorado University decision diagram (CUDD) (http://vlsi.colorado.edu/~fabio/CUDD/) and EXTRA (http://www.ee.pdx.edu/~alanmi/research/extra.htm) for the implementation of the operator $\otimes$ defined in Section V-B. For comparison, we also developed an implementation of our method without using the ZBDDs. Table V shows the algorithms used in our experiments. Table VI lists the parameters used for each experiment presented in this section.

A. Synthetic Data Sets

To verify the correctness of our method, we tested it with synthetic data sets that have predefined embedded patterns. The synthetic data were prepared as follows. We first created null matrices of 100 rows and five different numbers of columns

TABLE V
PATTERN-MINING METHODS TESTED IN THE EXPERIMENTS

ID	Name/description	Algorithm employed	Pattern types supported
METHOD 1	δ-biclustering [9]	Greedy iterative search	Type 3
METHOD 2	δ-pClustering [42]	Exhaustive enumeration	Type 3
METHOD 3	GEMS [43]	Gibbs sampling	Types 1, 3
METHOD 4	Our method implemented without ZBDDs	Algorithms 4, 5	Types 1, 2, 3
OUR METHOD	Our method fully implemented	Algorithms 4, 5	Types 1, 2, 3

TABLE VI
ALGORITHM PARAMETERS USED FOR THE EXPERIMENTS

Experiments		Algorithms and parameters								
Figure	Data	OUR METHOD/METHOD 4		METHOD 1		METHOD 2			METHOD 3	
		Type	τ	α	δ	δ	nr	nc	a	w
21	Synthetic	3	0	1.2	1000	0	10	10–120	0.01	250
23(a), 24(a), 26(a)	Yeast [10, 39]	3	75–80	1.2	300	10–15	80	10–12	0.25	15–30
22, 23(b), 24(b), 26(b)	Kidney [18]	3	20–25	1.2	150	20–25	10–14	18–20	0.01–0.1	25–75

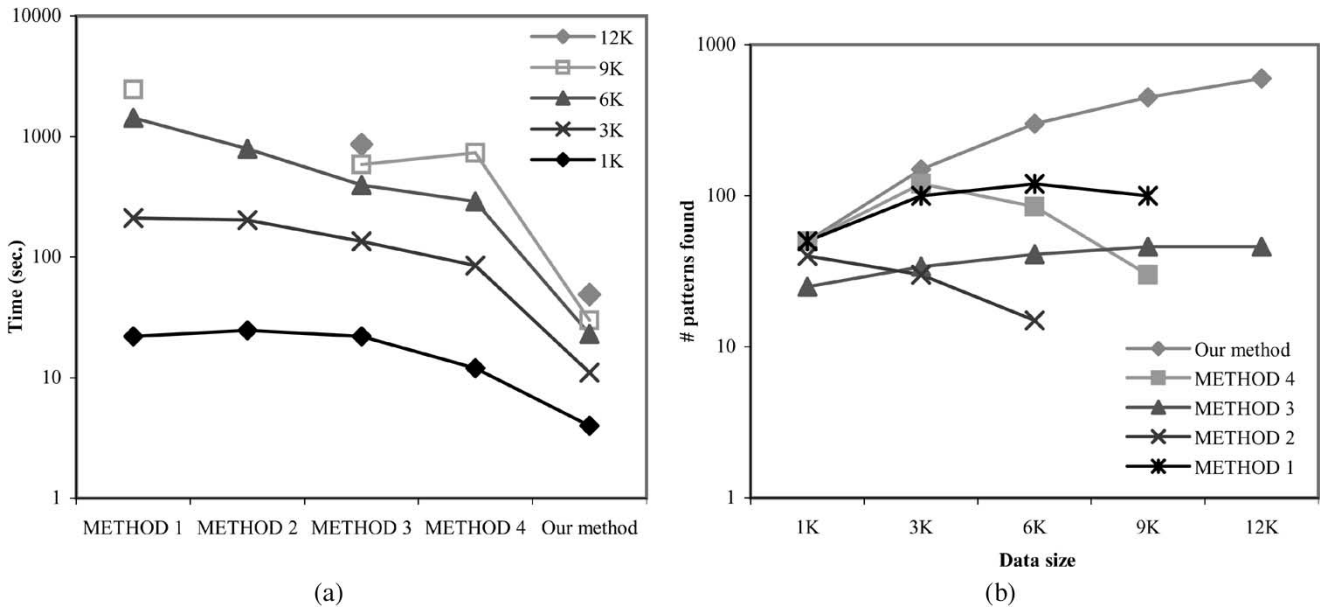


Fig. 21. Performance comparison using synthetic data sets. The missing points on the plots mean that the corresponding experiment could not be finished in reasonable time. (a) Response time spent by each method in order to find all the embedded patterns from the synthetic data sets of various sizes. (b) Number of patterns found by each method within the same time spent as our method.

(1K, 3K, 6K, 9K, and 12K). We then replaced the elements of each matrix with random numbers ranging from 0 to 500. For the matrix of $n = 100$ rows and $m \in \{1\text{K}, 3\text{K}, 6\text{K}, 9\text{K}, 12\text{K}\}$ columns, we embedded $0.05m$ predefined patterns that have at least $0.1n$ rows and at least $0.01m$ columns. Each predefined pattern was created in such a way that the values in every row or column fluctuate in harmony[3] and that all the methods involved in the experiment can detect it.

We invoked the methods listed in Table V with the parameters specified in Table VI. The results are depicted in Fig. 21. Fig. 21(a) shows the response time spent by each method in order to find all the embedded patterns. Fig. 21(b) shows the plot of the total number of patterns discovered by each method given the same time as that spent by our method implemented with ZBDDs. We can see in the experiments that it takes less time for our method to find all the embedded patterns and that our method can find more patterns given the same time, compared with the other methods tested. Especially, we observed that the use of ZBDDs indeed provides a substantial speed-up over the alternative implementation without ZBDDs.

[3]Every row or column is a shifted version of each other; examples are shown in Fig. 25(a) and (b).

B. Biological Data Sets

We tested our methods and the alternatives with a couple of large-scale data set obtained from actual biological experiments. Specifically, we used the gene-expression data sets produced by Affymetrix gene chips and cDNA microarrays, since this type of data is one of the largest and most widely available. As previously emphasized, our method is applicable to other types of data as well, as long as they can be represented by a matrix of real numbers. For more information on gene-expression data, we refer the interested reader to [20].

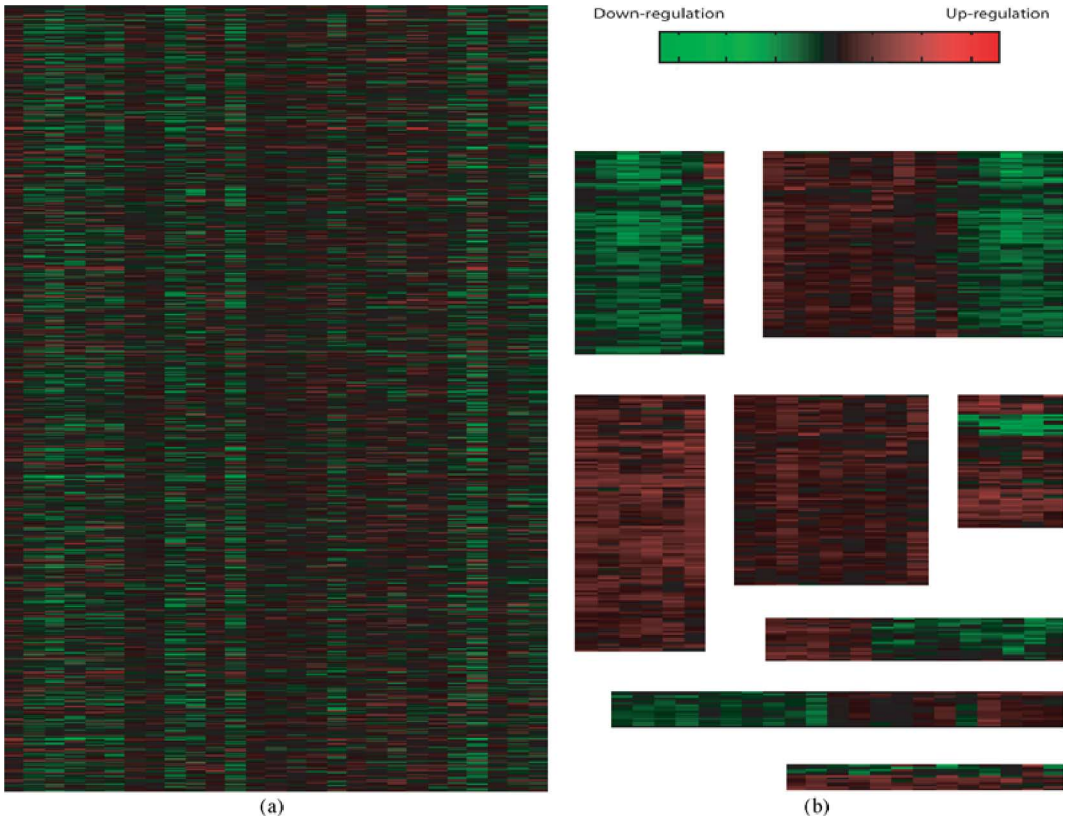


Fig. 22. Heat map of the renal-cell-carcinoma data [18] and some patterns found by our method. The legend for the heat map is also presented in the upper right corner. The red color indicates upregulation whereas the green color represents downregulation. The black color means no change in the regulation level. (a) The entire data matrix with 1876 rows (genes) and 27 columns (experimental conditions). (b) Some patterns (submatrices) discovered by our method.

1) Data Preparation: We used two different data sets. The first was the yeast *Saccharomyces cerevisiae* cell-cycle expression data [10], [39] produced by Affymetrix gene-chip experiments. This data set contains the expression information of 2884 genes under 17 experimental conditions. The second was the cDNA microarray data for renal-cell carcinoma [18], which represents the expression levels of 1876 genes under 27 different experimental conditions. Usually, gene-expression data is arranged in a data matrix, in which each row corresponds to one gene and each column to one experimental condition. Fig. 22 shows the heat map of this data set and some patterns found by our method.

2) Running-Time Comparison: We ran the methods listed in Table V with the parameters specified in Table VI. In the plots in Fig. 23(a) and (b), we compared the time to find the first k patterns from the yeast-cell-cycle data and the renal-cell-carcinoma data, respectively. The x-axis is the number of patterns produced and the y-axis is the response time to find these patterns. Our methods as well as METHOD 2 and METHOD 4 (see Table V) do not take as input the exact number of patterns to find. Thus, we ran these algorithms multiple times with different parameter values to find approximately k patterns. For METHOD 1 and METHOD 3, the exact number of patterns to find was specified as input parameters.

3) Pattern-Quality Evaluation: The experiments presented so far have demonstrated that our method outperforms the alternatives in terms of efficiency and the number of patterns. Here, we present more experimental results to show that our method can produce statistically more significant and biologically more-meaningful patterns, thus suggesting that our method can be helpful to the researchers in biomedicine as well. To this end, we utilize the concept of correspondence plots [38] and the mean-squared-residue (MSR) scores [9].

Correspondence plot. To assess the statistical significance and biological meaning of discovered patterns, we employed a technique [38] that enables us to compute the p-value of each pattern with respect to known (putatively correct) biological knowledge. Suppose prior knowledge classifies N genes into M classes, $H_1, H_2, \ldots, H_M$. Let P be a pattern with g genes and assume that out of those g genes, g_j genes belong to class H_j. Assuming the most abundant class for the genes in P is H_i, the hypergeometric distribution is used to calculate p, the p-value of the pattern P

$$p = \sum_{k=g_i}^{g} \frac{\binom{|H_i|}{k}\binom{N-|H_i|}{g-k}}{\binom{N}{g}}. \tag{16}$$

That is, the p-value corresponds to the probability of obtaining at least g_i elements of the class H_i in a random set of size g. As the known biological knowledge, the categories of yeast genes proposed by Tavazoie *et al.* [39] and the human-genes classes reported by Higgins *et al.* [18] were used for our experiments.

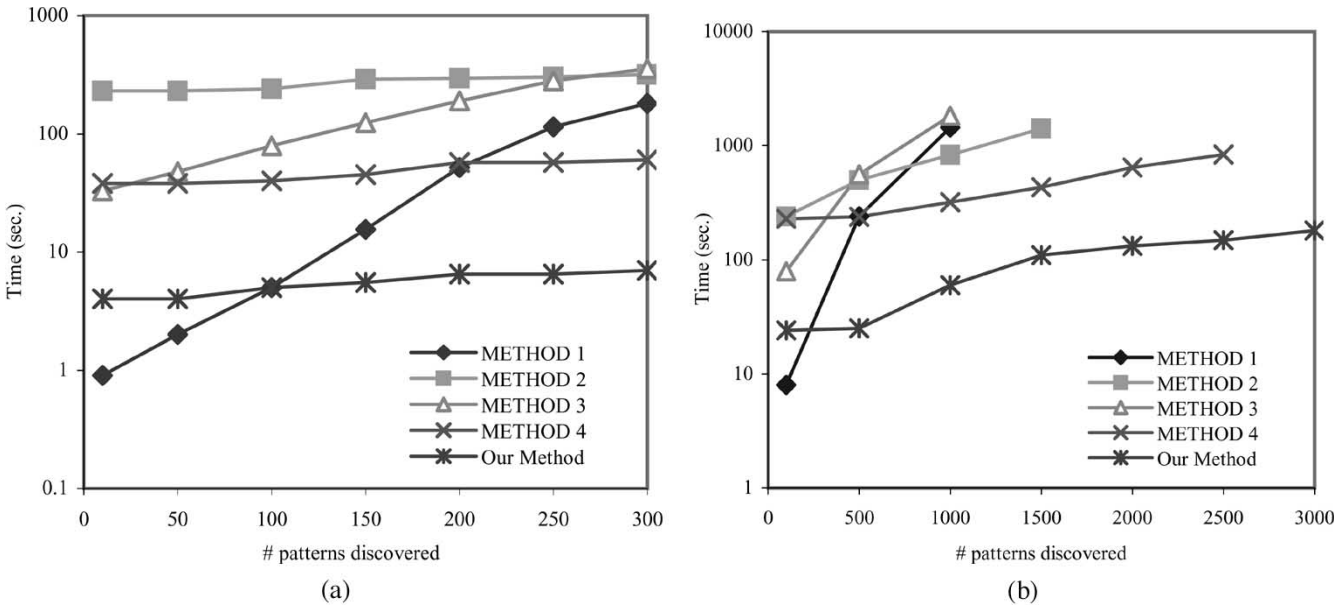


Fig. 23. Performance comparison using biological data sets. The missing points on the plots mean that the corresponding experiment could not be finished in reasonable time. (a) Yeast-cell-cycle data [10], [39]. (b) Renal-cell carcinoma [18].

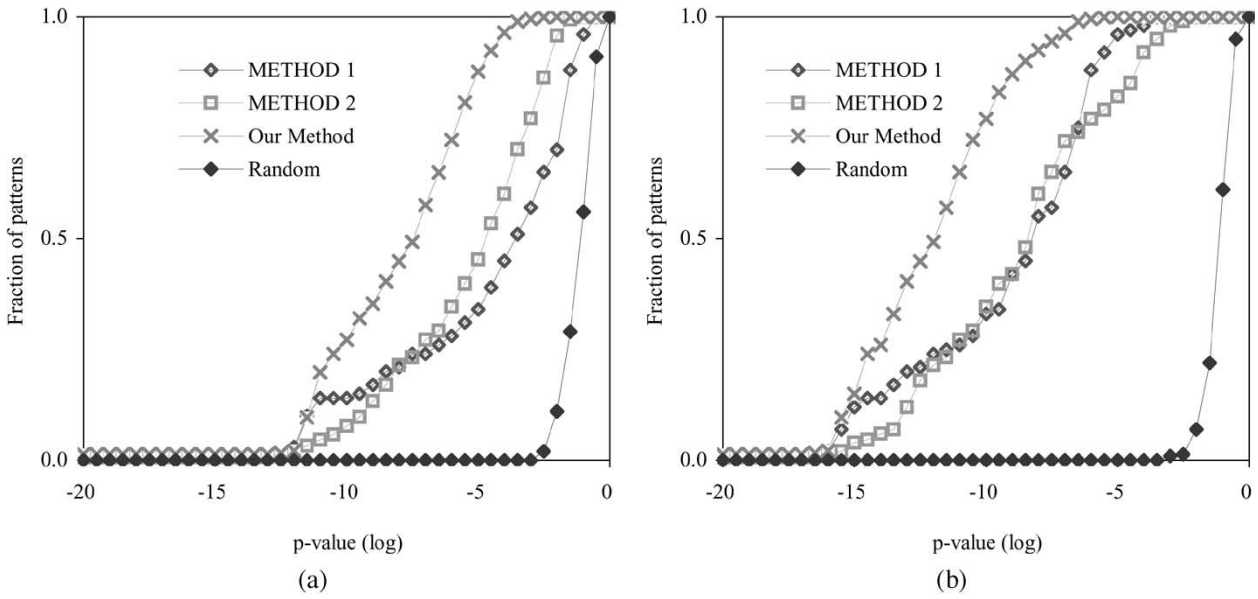


Fig. 24. Correspondence plots that show the distribution of p-values of the produced patterns with respect to prior biological knowledge. A point (x, y) on the plot presents the fraction (y) of patterns whose p-value is at most x. (a) Yeast-cell cycle [39]. (b) Renal-cell carcinoma [18].

In the correspondence plot, early departure of a curve from the x-axis indicates the existence of patterns with low p-values. Consequently, the area under a curve approximately shows the degree of statistical significance of the patterns used to draw the curve.

Fig. 24 presents the correspondence plots for the patterns generated by several different methods on the yeast data and the renal-cell-carcinoma data. The plots also include randomly generated patterns. Both plots indicate that the patterns shown are all far from the random noise. It is also demonstrated that the patterns found by our algorithm tend to be more statistically significant than the others, meaning that our patterns conform to the known biological classification more accurately.

MSR scores. The MSR sores can measure the degree of coherence exhibited by the elements in a matrix [9]. In the analysis of variance (ANOVA), a residue is defined for each element in a matrix as the difference between the element and the mean of all elements of the matrix [32]. The residue of element a_{ij} of a matrix denoted by pair (I, J) is $r_{ij} = a_{ij} - \overline{a}_{i\bullet} - \overline{a}_{\bullet j} + \overline{a}_{\bullet\bullet}$, where $\overline{a}_{i\bullet}$ is the mean of the ith row, $\overline{a}_{\bullet j}$ the mean of the jth column, and $\overline{a}_{\bullet\bullet}$ is the mean of all elements in A. The MSR of the matrix is then defined as

$$\text{MSR}(I, J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} r_{ij}^2. \tag{17}$$

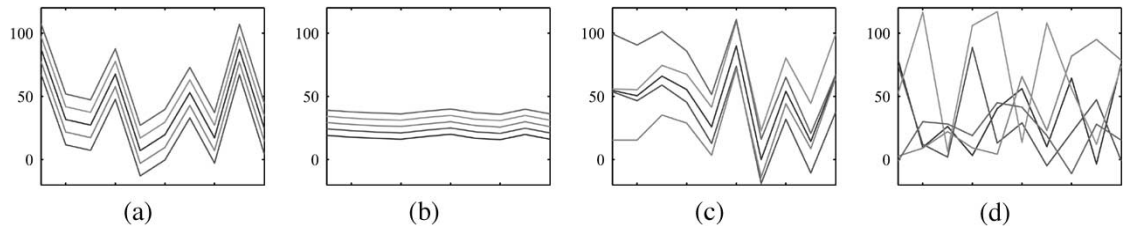


Fig. 25. MSR scores as a measure of pattern quality. A low MSR value typically means a high level of coherence, and *vice versa* [9]. (a) MSR $= 0$, (b) MSR $= 0$, (c) MSR $= 103$, and (d) MSR $= 946$.

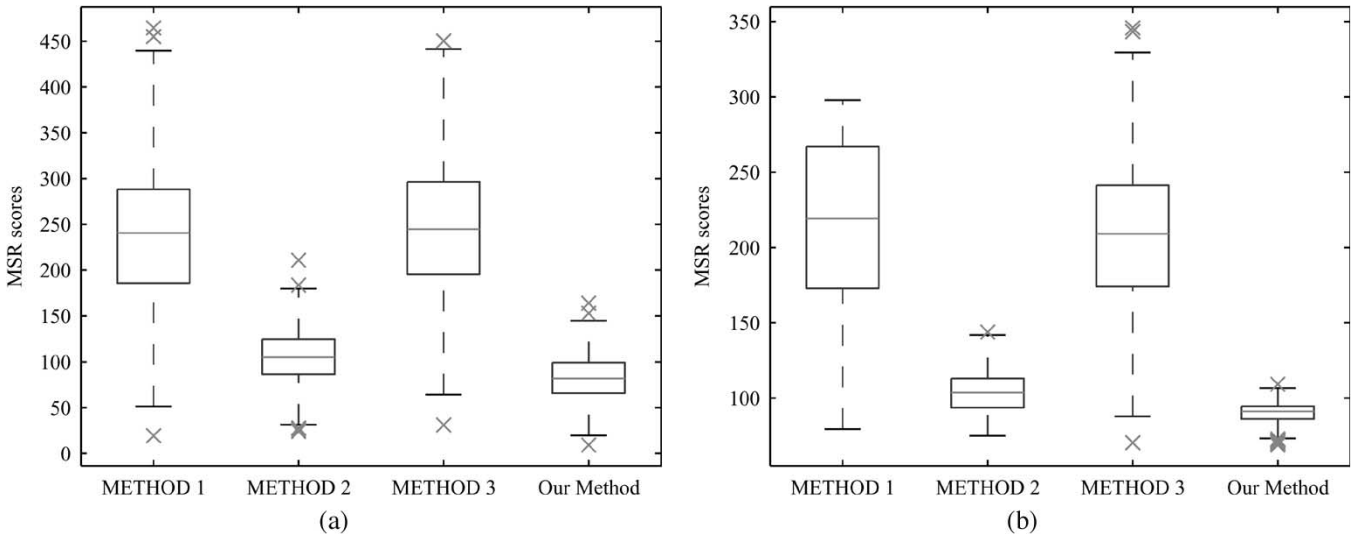


Fig. 26. Box plots for MSR comparison. The line in the middle of a box indicates the position of the median. The upper and lower boundaries of the box represent the location of the 75th and 25th percentiles, respectively. The symbol "$\times$" outside the ends of the tails corresponds to outliers. (a) Yeast-cell cycle [39]. (b) Renal-cell carcinoma [18].

Thus, a low value of residue typically means a high level of coherence, and vice versa [9]. For example, the MSR score of the patterns depicted in Fig. 25(a) and (b) is zero, since the values fluctuate in harmony. In contrast, the pattern shown in Fig. 25(d) is very noisy, and thus, has a higher MSR score. The pattern in Fig. 25(c) has an intermediate MSR score. Consequently, the MSR scores can be useful to evaluate the quality of patterns of all types defined in this study.

Fig. 26(a) and (b) shows box plots to compare the MSR scores of the patterns discovered from the yeast-cell cycle and the renal-cell-carcinoma data, respectively. A box plot is a plot that represents graphically several descriptive statistics such as the median and percentiles of a data sample [14]. The reader can refer to the caption of Fig. 26 to find how to read a box plot.

As is evident from the box plots, the patterns found by our method have the lowest median MSR scores in our experiments. To quantitatively establish this observation, we performed the Wilcoxon rank-sum test [14] to compare the patterns discovered by our method with the others. We generated approximately 500 patterns per method for each data set and compared a group of patterns found by our method with another group of patterns detected by an alternative method. In all the cases we tested, the difference in the median was statistically significant at 0.01% level $(P < 0.0001)$. This result shows that our method tends to find better patterns with respect to the MSR scores.

VII. Conclusion

Compared with conventional biological data-acquisition techniques, better productivity, reliability, and speed are possible through the miniaturization and integration realized in microfluidics-based biochips. Given that the throughput of these fascinating technologies is growing fast, it is crucial to have efficient computational tools to analyze the large-scale biological data obtained. In this paper, we proposed an effective pattern-mining method that can be useful for a variety of biochip applications. Given a data matrix, the proposed method can find patterns appearing as a submatrix of the data matrix. In particular, we introduced the notion of homogeneous patterns and formulated the problem of finding three types of homogeneous patterns frequently encountered in the literature. We also mathematically characterized the problem and developed a novel method applicable to large-scale biological data. The proposed method employed dynamic programming as well as efficient data structures such as ZBDDs, which were particularly useful to extend the scalability of our method. Consequently, given a data matrix of practical scale, our approach can find with great efficiency all the homogeneous patterns that

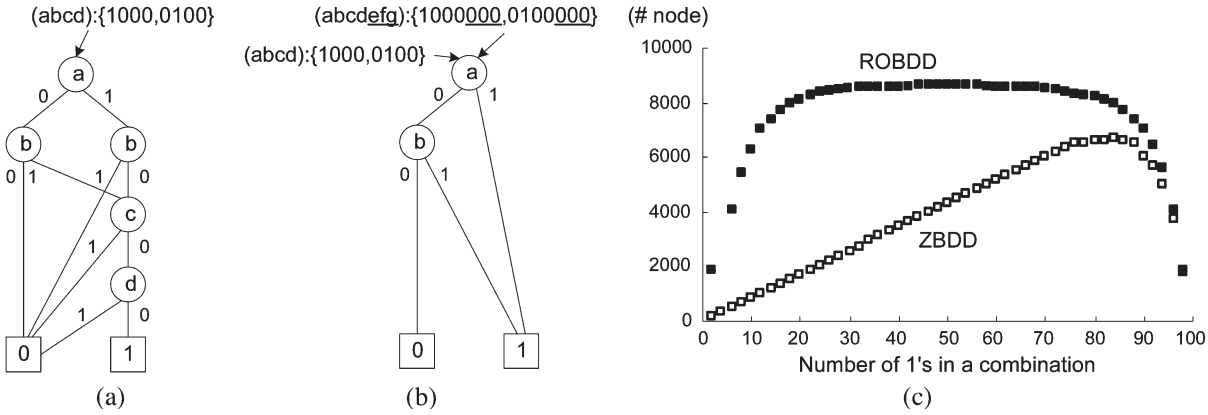


Fig. 27. Representation of a set of combinations. (a) ROBDD representation. (b) ZBDD representation. (c) Comparison of ROBDD and ZBDD [27].

satisfy specific input parameters. We tested our method with the biochip data produced by Affymetrix gene chips and cDNA microarrays and confirmed the effectiveness of our approach. Therefore, we conclude that our method can provide the designers of high-throughput biochips with the necessary feedback for the next design iteration in a timely manner.

Appendix A
Zero-Suppressed Binary Decision Diagrams (ZBDDs)

In most combinatorial applications, sets of combinations (see Section V-B3) are sparse, which are defined as follows [25].

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

The ZBDD [26], [27] is an efficient data structure to represent and manipulate a set of combinations. Minato [26], [27] proposed two reduction rules to reduce ordinary BDDs to ZBDDs: 1) merge equivalent subgraphs; and 2) if the 1-edge of a node v points to the 0-terminal vertex, then eliminate v and redirect all incoming edges of v to the 0-successor of v. Consequently, ZBDDs can exploit both types of sparsity defined above and provide an efficient representation for manipulating large-scale sets of combinations [25].

For instance, the ROBDD in Fig. 27(a) represents a set of combinations {1000, 0100} for four input variables $(abcd)$. Each path from the root vertex to the 1-leaf corresponds to a combination. By applying the ZBDD reduction rules, we can reduce the BDD in Fig. 27(a) to the ZBDD in Fig. 27(b), which is more compact in terms of the number of vertices. As shown in Fig. 27(c), Minato [27] compared the size of a ZBDD with that of an ROBDD for a large set of combinations and showed that ZBDDs provide a much more compact representation of sets of combinations in most cases.

ZBDD representations are independent of the number of input variables as long as the combination remains the same, which is due to the "zero-suppression" effect. Consequently, we do not need to fix the number of input variables before generating graphs, and ZBDDs automatically suppress the variables that never appear in any combination [27]. For example, a set of combinations {1000000, 0100000} for seven variables $(abcdefg)$ is represented by the same ZBDD in Fig. 27(b). This property does not hold for other types of BDDs.

Appendix B
Proof of Theorem 1

We can prove the theorem by showing that: 1) $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \supseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$; and 2) $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \subseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$.

Proof: We first prove (1). For the sake of contradiction, assume that $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \subset \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$. This means that there exists a set S such that $S \in \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$ and $S \notin (\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W}$. Assume that $S = T \cap (U \cap W)$, where $T \in \mathcal{T}$, $U \in \mathcal{U}$, and $W \in \mathcal{W}$. Since $S \notin (\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W}$, there must exist $W' \in \mathcal{W}$ such that $(T \cap U) \cap W \subset (T \cap U) \cap W'$. By the associative law for basic set intersection, $T \cap (U \cap W) = (T \cap U) \cap W \subset (T \cap U) \cap W' = T \cap (U \cap W')$. In other words, if $S \notin (\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W}$, then there must exist $W' \in \mathcal{W}$ such that $(U \cap W) \subset (U \cap W')$. However, since $S \in \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$, there cannot exist $W' \in \mathcal{W}$ such that $(U \cap W) \subset (U \cap W')$. We have reached a contradiction, and thus our original assumption that $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \subset \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$ must be false. Therefore, $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \supseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$. By symmetry, we can prove $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \subseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$ in a similar way.

We have shown that $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \supseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$ and $(\mathcal{T} \otimes \mathcal{U}) \otimes \mathcal{W} \subseteq \mathcal{T} \otimes (\mathcal{U} \otimes \mathcal{W})$, which completes the proof. ■

Appendix C
Proof of Theorem 2

The derivation of (4)–(6) is straightforward from the definition of atomic patterns. If the set I has only one row (type 1) or two (types 2 and 3), the image $\mathfrak{J}(I)$ simply consists of the column set of atomic patterns for the row(s) in I. Equations (7) and (8) can be derived from the generalization of Properties 1 and 2, respectively, by replacing the operator $\cap$ with the operator $\otimes$ defined in Section V-B.

Here, we focus on the derivation of (9). To this end, we first propose the following lemma.

Lemma 1: Let (I, J) be a homogeneous pattern. If $\{i, k\} \subseteq I$, then there exists at least one set $J' \in \mathfrak{J}_3(\{i, k\})$ such that $J \subseteq J'$.

Proof: Assume $J \supset J'$ for all $J' \in \mathfrak{J}_3(\{i, k\})$. Since (I, J) is a homogeneous pattern and $I \supseteq \{i, k\}$, its subpattern

$(\{i,k\}, J)$ is also a homogeneous pattern under the same definition. By definition, if $J' \in \mathfrak{J}_3(\{i,k\})$, then there exists no $J'' \supset J'$ such that $(\{i,k\}, J'')$ is yet another homogeneous pattern under the same definition. We have reached a contradiction, and thus our original assumption that $J \supset J'$ for all $J' \in \mathfrak{J}_3(\{i,k\})$ must be false. Therefore, there must be at least one instance of $J' \in \mathfrak{J}_3(\{i,k\})$ such that $J \subseteq J'$.

Now we derive (9). Let $P = (I, J)$ be a maximal homogeneous pattern. Then, by Lemma 1, for each $\{i,k\} \subseteq I$, there exists at least one set $J_{\{i,k\}} \in \mathfrak{J}_3(\{i,k\})$ such that $J \subseteq J_{\{i,k\}}$. For the sake of explanation, assume for now that only one such $J_{\{i,k\}}$ is contained in each $\mathfrak{J}_3(\{i,k\})$. Then, it follows that:

$$J \subseteq \bigcap_{\forall\{i,k\}\subseteq I} J_{\{i,k\}}. \tag{18}$$

Moreover, since the pattern P is maximal, there is no J' such that $J' \supset J$ and $J' \subseteq \bigcap_{\forall\{i,k\}\subseteq I} J_{\{i,k\}}$. Thus, the following equation holds for J:

$$J = \bigcap_{\forall\{i,k\}\subseteq I} J_{\{i,k\}}. \tag{19}$$

In general, each $\mathfrak{J}_3(\{i,k\})$ can have multiple instances of $J_{\{i,k\}}$, not only one as previously assumed. Thus, we can have multiple instances of (19), which can be compactly represented using the operator $\otimes$ defined in Section V-B

$$J \in \bigotimes_{\forall\{i,k\}\subseteq I} \left\{ J_{\{i,k\}} | J_{\{i,k\}} \in \mathfrak{J}_3\left(\{i,k\}\right), J \subseteq J_{\{i,k\}} \right\}. \tag{20}$$

Finally, suppose that we replace the operands of $\otimes$ in (20) with $\{J_{\{i,k\}} | J_{\{i,k\}} \in \mathfrak{J}_3(\{i,k\})\} = \mathfrak{J}_3(\{i,k\})$, removing the constraint on J. Then, we can find not only the set J but also the other column sets that can form a homogeneous pattern with the row set I

$$\{\text{all column sets that can form a type-3 pattern with } I\} = \bigotimes_{\forall\{i,k\}\subseteq I} \mathfrak{J}_3\left(\{i,k\}\right). \tag{21}$$

By definition, the operator $\otimes$ gives only maximal sets. Therefore, (21) is equivalent to (9). We have derived (9), and this completes the proof of Theorem 2. ■

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers of the manuscript for their valuable comments. The authors gratefully acknowledge E. Ficarra, C. Nardini, and Dr. A. Mishchenko for helpful discussions.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.

[2] R. B. Altman and S. Raychaudhuri, "Whole-genome expression analysis: Challenges beyond clustering," *Curr. Opin. Struct. Biol.*, vol. 11, no. 3, pp. 340–347, Jun. 2001.

[3] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini,"Discovering local structure in gene expression data: The order-preserving submatrix problem," *J. Comput. Biol.*, vol. 10, no. 3–4, pp. 373–384, Aug. 2003.

[4] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. 27th Design Automation Conf.*, Orlando, FL, 1990, pp. 40–45.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[6] ——, "Binary decision diagrams and beyond: Enabling technologies for formal verification," in *IEEE/ACM Int. Conf. Computer Aided Design (ICCAD)*, San Jose, CA, 1995, pp. 236–243.

[7] A. Califano, G. Stolovitzky, and Y. Tu, "Analysis of gene expression microarrays for phenotype classification," in *Proc. Int. Conf. Intelligent Systems Molecular Biology*, San Diego, CA, 2000, pp. 75–85.

[8] W. Chen, M. Reiss, and D. J. Foran, "A prototype for unsupervised analysis of tissue microarrays for cancer research and diagnostics," *IEEE Trans. Inf. Technol. Biomed.*, vol. 8, no. 2, pp. 89–96, Jun. 2004.

[9] Y. Cheng and G. M. Church, "Biclustering of expression data," in *Proc. Intelligent Systems Molecular Biology (ISMB)*, San Diego, CA, 2000, pp. 93–103.

[10] R. J. Cho, M. J. Campbell, E. A. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T. G. Wolfsberg, A. E. Gabrielian, D. Landsman, D. J. Lockhart, and R. W. Davis, "A genome-wide transcriptional analysis of the mitotic cell cycle," *Mol. Cell*, vol. 2, no. 1, pp. 65–73, Jul. 1998.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

[12] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[13] J. DeRisi, L. Penland, P. O. Brown, M. L. Bittner, P. S. Meltzer, M. Ray, Y. Chen, Y. A. Su, and J. M. Trent, "Use of a cDNA microarray to analyse gene expression patterns in human cancer," *Nat. Genet.*, vol. 14, no. 4, pp. 457–460, Dec. 1996.

[14] S. Drăghici, *Data Analysis Tools for DNA Microarrays*. Boca Raton, FL: CRC, 2003.

[15] A. C. R. Grayson, R. S. Shawgo, A. M. Johnson, N. T. Flynn, Y. Li, M. J. Cima, and R. Langer, "A BioMEMS review: MEMS technology for physiologically integrated devices," *Proc. IEEE*, vol. 92, no. 1, pp. 6–21, Jan. 2004.

[16] D. Gusfield, *Algorithms on String, Trees and Sequences: Computer Science and Computational Biology*. New York: Cambridge Univ. Press, 1997.

[17] A. Hassibi and T. H. Lee, "A programmable electrochemical biosensor array in 0.18 μm standard CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, Feb. 2005, pp. 564–566.

[18] J. P. T. Higgins, R. Shinghal, H. Gill, J. H. Reese, M. Terris, R. J. Cohen, M. Fero, J. R. Pollack, M. van de Rijn, and J. D. Brooks, "Gene expression patterns in renal cell carcinoma assessed by complementary DNA microarray," *Amer. J. Pathol.*, vol. 162, no. 3, pp. 925–932, Mar. 2003.

[19] S. Kiyonaka, K. Sada, I. Yoshimura, S. Shinkai, N. Kato, and I. Hamachi, "Semi-wet peptide/protein array using supramolecular hydrogel," *Nat. Mater.*, vol. 3, no. 1, pp. 58–64, Jan. 2004.

[20] I. S. Kohane, A. T. Kho, and A. J. Butte, *Microarrays for an Integrative Genomics*. Cambridge, MA: MIT Press, 2003.

[21] J. Liu, J. Yang, and W. Wang, "Biclustering in gene expression data by tendency," in *Proc. Computational Systems Bioinformatics Conf. (CSB)*, Stanford, CA, 2004, pp. 182–193.

[22] D. Lockhart, H. Dong, M. Byrne, M. Follettie, M. Gallo, M. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Horton, and E. L. Brown, "Expression monitoring by hybridization to high-density oligonucleotide arrays," *Nat. Biotechnol.*, vol. 14, no. 13, pp. 1675–1680, Dec. 1996.

[23] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 1, no. 1, pp. 24–45, Jan.–Mar. 2004.

[24] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.

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

[26] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in *IEEE/ACM Design Automation Conf. (DAC)*, Dallas, TX. New York: ACM Press, 1993, pp. 272–277.

[27] ——, *Binary Decision Diagrams and Applications for VLSI CAD*. Norwell, MA: Kluwer, 1996.

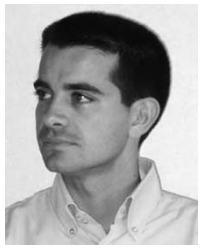
[28] S. Minato and H. Arimura, "Combinatorial item set analysis based on zero-suppressed BDDs," Hokkaido Univ., Sapporo, Japan, Tech. Rep. TCS-TR-A-04-1, Dec. 2004.

[29] T. M. Murali and S. Kasif, "Extracting conserved gene expression motifs

from gene expression data," in *Proc. Pacific Symp. Biocomputing*, Kona, HI, 2003, pp. 77–88.
[30] R. Peeters, "The maximum edge biclique problem is NP-complete," *Discrete Appl. Math.*, vol. 131, no. 3, pp. 651–654, Sep. 2003.
[31] S. Raychaundhuri, P. D. Sutphin, J. T. Chang, and R. B. Altman, "Basic microarray analysis: Grouping and feature reduction," *Trends Biotech.*, vol. 19, no. 5, pp. 189–193, May 2001.
[32] J. A. Rice, *Mathematical Statistics and Data Analysis*. Belmont, CA: Duxbury Press, 1994.
[33] A. Y. Rubina, E. I. Dementieva, A. A. Stomakhin, E. L. Darii, S. V. Pan'kov, V. E. Barsky, S. M. Ivanov, E. V. Konovalova, and A. D. Mirzabekov, "Hydrogel-based protein microchips: Manufacturing, properties, and applications," *Biotechniques*, vol. 34, no. 5, pp. 1008–1014, May 2003.
[34] T. Sasao and M. Fujita, *Representations of Discrete Functions*. Norwell, MA: Kluwer, 1996.
[35] M. Schienle, C. Paulus, A. Frey, F. Hofmann, B. Holzapfl, P. Schindler-Bauer, and R. Thewes, "A fully electronic DNA sensor with 128 positions and in-pixel A/D conversion," *IEEE J. Solid-State Circuits*, vol. 39, no. 12, pp. 2438–2445, Dec. 2004.
[36] E. Scrivener, R. Barry, A. Platt, R. Calvert, G. Masih, P. Hextall, M. Soloviev, and J. Terrett, "Peptidomics: A new approach to affinity protein microarrays," *Proteomics*, vol. 3, no. 2, pp. 122–128, Feb. 2003.
[37] I. S. Shergill, N. K. Shergill, M. Arya, and H. R. Patel, "Tissue microarrays: A current medical research tool," *Curr. Med. Res. Opin.*, vol. 20, no. 5, pp. 707–712, May 2004.
[38] A. Tanay, R. Sharan, and R. Shamir, "Discovering statistically significant biclusters in gene expression data," *Bioinformatics*, vol. 18, no. 90001, pp. S136–S144, Jul. 2002.
[39] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and G. M. Church, "Systematic determination of genetic network architecture," *Nat. Genet.*, vol. 22, no. 3, pp. 281–285, Jul. 1999.
[40] E. Verpoorte and N. F. de Rooij, "Microfluidics meets MEMS," *Proc. IEEE*, vol. 91, no. 6, pp. 930–953, Jun. 2003.
[41] C. C. Wang, R. P. Huang, M. Sommer, H. Lisoukov, R. Huang, Y. Lin, T. Miller, and J. Burke, "Array-based multiplexed screening and quantitation of human cytokines and chemokines," *J. Proteome Res.*, vol. 1, no. 4, pp. 337–343, Jul./Aug. 2002.
[42] H. Wang, W. Wang, J. Yang, and P. S. Yu, "Clustering by pattern similarity in large data sets," in *Proc. ACM SIGMOD Int. Conf. Management Data*, Madison, WI, 2002, pp. 394–405.
[43] C.-J. Wu, Y. Fu, T. M. Murali, and S. Kasif, "Gene expression module discovery using Gibbs sampling," *Genome Inform.*, vol. 15, no. 1, pp. 239–248, 2004.
[44] J. Yang, H. Wang, W. Wang, and P. Yu, "Enhanced biclustering on expression data," in *Proc. IEEE 3rd Symp. Bioinformatics and Bioengineering*, Washington, DC, 2003, pp. 321–327.
[45] S. Yoon, C. Nardini, L. Benini, and G. De Micheli, "An application of zero-suppressed binary decision diagrams to clustering analysis of DNA microarray data," in *Proc. 26th Annu. Int. Conf. IEEE EMBS*, San Francisco, CA, Sep. 2004, pp. 2925–2928.
[46] ——, "Enhanced pClustering and its applications to gene expression data," in *Proc. IEEE 4th Symp. Bioinformatics and Bioengineering*, Taichung, Taiwan, May 2004, pp. 275–282.

Sungroh Yoon received the B.S. degree in electrical engineering from Seoul National University, Korea, in 1996 and the M.S. degree and Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2002 and 2005, respectively.

He has research interests in computer-aided design and analysis of high-throughput biochip systems and machine-learning applications in computational biology and bioinformatics.



Luca Benini (S'94–M'97–SM'04) received a Ph.D. degree in electrical engineering from Stanford University, Stanford, CA in 1997.

He is an Associate Professor at the Department of Electrical Engineering and Computer Science (DEIS), University of Bologna, Bologna, Italy. His research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems. On these topics, he has published more than 250 papers in international journals and conferences and three books.

Dr. Benini has been Program Chair and Vice-chair of the Design Automation and Test in Europe Conference. He is a member of the Technical Program Committee and Organizing Committee of several technical conferences, including the Design Automation Conference, International Symposium on Low Power Design, and the Symposium on Hardware–Software Codesign.

Giovanni De Micheli (S'79–M'79–SM–89–F'94) received the B.S. degree in nuclear engineering from the Politecnico di Milano, Milan, Italy, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California at Berkeley in 1980 and 1983, respectively.

He is a Professor and the Director of the Integrated Systems Centre at EPF Lausanne, Switzerland, and the President of the Scientific Committee of CSEM, Neuchatel, Switzerland. Previously, he was a Professor of Electrical Engineering at Stanford University. His research interests include several aspects of design technologies for integrated systems on silicon, such as synthesis, hardware–software (hw/sw) codesign, and low-power design, as well as systems on heterogeneous platforms including electrical, optical, micromechanical, and biological components. He is the author of *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994), coauthor and/or coeditor of five other books and of over 300 technical articles. He is, or has been, a member of the Technical Advisory Board of several companies, including Magma Design Automation, Coware, Aplus Design Technologies, IROC, Ambit Design Systems, and STMicroelectronics.

Dr. De Micheli is the recipient of the 2003 IEEE Emanuel Piore Award for contributions to computer-aided synthesis of digital systems. He is a fellow of the Association for Computing Machinery (ACM). He received the Golden Jubilee Medal for outstanding contributions to the IEEE Circuits and Systems (CAS) Society in 2000. He received the 1987 D. Pederson Award for the Best Paper on the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, two Best Paper Awards at the Design Automation Conference, in 1983 and in 1993, and a Best Paper Award at the Design, Automation and Test in Europe (DATE) Conference in 2005. He was President of the IEEE CAS Society in 2003. He was Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER AND DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS in 1987–2001. He was the Program Chair and General Chair of the Design Automation Conference (DAC) in 1996–1997 and 2000, respectively. He was the Program and General Chair of the International Conference on Computer Design (ICCD) in 1988 and 1989, respectively.