# Mining Frequent Closed Cubes in 3D Datasets

Liping Ji Kian-Lee Tan Anthony K. H. Tung
National University of Singapore
Department of Computer Science
3 Science Drive 2, Singapore 117543

## ABSTRACT

In this paper, we introduce the concept of frequent closed cube (FCC), which generalizes the notion of 2D frequent closed pattern to 3D context. We propose two novel algorithms to mine FCCs from 3D datasets. The first scheme is a Representative Slice Mining (RSM) framework that can be used to extend existing 2D FCP mining algorithms for FCC mining. The second technique, called CubeMiner, is a novel algorithm that operates on the 3D space directly. We have implemented both schemes, and evaluated their performance on both real and synthetic datasets. The experimental results show that the RSM-based scheme is efficient when one of the dimensions is small, while CubeMiner is superior otherwise.

## 1. INTRODUCTION

Frequent pattern mining plays an important role in many data mining tasks, such as association rule analysis [1], sequential patterns [2], episodes [7], partial periodicity [5], and etc. However, frequent pattern (FP) mining is a time-consuming process. Moreover, it may generate too many patterns and rules (a large number of which are "redundant" in the sense that they do not shed additional insights) for users to digest. To overcome these problems, Pasquier et. al. proposed [10] the notion of frequent closed pattern (FCP). While the number of FCPs are much smaller than the FPs, they carry the same information as the FPs.

Several efficient FCP mining algorithms have been proposed in the literature, including feature enumeration algorithms [11, 17], row enumeration algorithms [8] and dense-data mining algorithms [3]. However, these algorithms are all limited to 2D dataset analysis, for example, the *gene-time*, *gene-sample* biological datasets in microarray dataset analysis, and the *transaction-itemset* datasets in 'market basket' analysis. With recent advances in microarray technology, the expression levels of a set of genes under a set of samples can be measured simultaneously over a series of time points, which results in 3D *gene-sample-time* microarray data [6]. Even in the traditional 'market-basket' analysis, it is not uncommon to have consumer information on a number of dimensions, e.g., *region-time-items* data that stores the sales of itemsets in certain locations over certain time periods. This trend motivates us to extend existing 2D frequent closed pattern analysis to 3D context. We refer to the frequent closed pattern in 3D context as *frequent closed cube* (FCC). Designing efficient algorithms to discover FCCs is the theme of this paper.

Association analysis based on FCCs can deliver more interesting information in 3D context. Let us first take biological microarray datasets for example. Association analysis based on FCCs can reveal patterns about how the expression of one gene may be associated with the expression of a set of genes under a set of environments during a set of time points. Given such information, we can easily infer that the genes involved participate in some kind of gene networks. Moreover, such association rules can be used to relate the expression of genes to their cellular environments and time periods simultaneously. Such associations can help to detect cancer genes in different cancer developing stages, especially when cancer is caused by a set of genes acting together instead of a single gene. Like clustering, gene function can be inferred based on the other genes in such association rule. Next, we give an example in 'market basket' analysis. While the association analysis based on 2D frequent pattern represents a set of items that is likely to be purchased together in a set of transactions, a 3D FCC over a sales (region-time-items) dataset would represent a set of items that is likely to be purchased together in several locations over a set of time periods. Such information would enable suppliers to deploy their products to chains located at different places during certain periods where consumers share similar purchasing behaviors.

In this paper, we tackle the problem of mining FCC from 3D datasets. Our contributions are as follows. First, we introduce the notion of FCC and formally define it. Second, we propose two approaches to mine FCCs. The first approach is a three-phase framework, called Representative Slice Mining algorithm (RSM) that exploits 2D FCP mining algorithms to mine FCCs. The basic idea is to transform a 3D dataset into a set of 2D datasets, mine the 2D datasets using an existing 2D FCP mining algorithm, and then prune away any frequent cubes that are not closed. The second method is a novel scheme, called CubeMiner, that operates directly on the 3D dataset to mine FCCs. Third, we also show how RSM and CubeMiner can be easily extended to exploit parallelism. Finally, we have implemented RSM and


*VLDB '06,* September 12-15, 2006, Seoul, Korea.

CubeMiner, and conducted experiments on both real and synthetic datasets. To our knowledge, there has been no prior work that mine FCCs.

The rest of this paper is organized as follows. Section 2 reviews some related works. In Section 3, we formally define the FCC mining problem. Section 4 presents the proposed RSM framework, while Section 5 presents the proposed CubeMiner algorithm. In Section 6, we show how RSM and CubeMiner can be extended to exploit parallelism. Section 7 reports experimental results on RSM and CubeMiner, and finally, we conclude in Section 8.

## 2. RELATED WORK

Traditionally, frequent pattern mining algorithms [1, 18, 14] typically generate a large number of patterns and many of them are redundant. To reduce the number of frequent patterns, frequent closed pattern (FCP) mining algorithms have been proposed. A-close [10] uses a breadth-first search to find FCPs. CLOSET [11] and CLOSET+ [16] adopt a depth-first, feature enumeration strategy. CLOSET uses a frequent pattern tree for a compressed representation of the dataset. CLOSET+, an enhanced version of CLOSET, uses a hybrid tree-projection method to build conditional projected table in two different ways according to the density of the dataset. Both MAFIA [4] and CHARM [17] use a vertical representation of the datasets. MAFIA adopts a compressed vertical bitmap structure while CHARM enumerates closed itemsets using a dual itemset-tidset search tree and adopts the *Diffset* technique to reduce the size of the intermediate tidsets. Since these methods adopt a feature enumeration strategy, they cannot efficiently handle datasets with a large number of features (columns).

More recently, several schemes have been designed to handle "large columns small rows" datasets. In [8], the scheme CARPENTER combines depth-first, row enumeration strategy with some efficient search pruning techniques. In [9], COBBLER dynamically switches between feature enumeration and row enumeration depending on the data characteristic in the process of mining. Both schemes, however, cannot handle dense datasets. In [3], D-miner was proposed to identify closed sets of attributes (or items) for dense and highly-correlated boolean contexts. D-miner generates and employs a set of cutters (containing "0" information) to divide the whole dataset into small dense spaces.

Although the above-mentioned algorithms perform well in their respective application domains in 2D datasets, they cannot mine FCCs in 3D context.

In [13], a scheme was proposed to discover calendric association rules. Although time intervals are taken as a third dimension, they are pre-defined by users as calendric information. Hence, no thorough enumeration on the third dimension is employed and no 'close' constraint is put on any dimension. In [12], sequential pattern mining is studied in multi-dimensional context. However, it is still 2D frequent pattern mining along with multi-dimensional projected database. The third or even the fourth dimensions do not fully enumerate on different entries as what the two base dimensions do, and different entries on the third/fourth dimension are only employed to divide the data records into different projected groups. Moreover, no "close" relationships between the third/fourth dimension and the two base dimensions are delivered. Thus, these works cannot be extended to mine FCCs.

More recently, [6] and [19] proposed clustering algorithms to analyze clusters on 3D microarray data, however, such algorithms cannot be employed to mine 3D frequent closed patterns.

In this paper, we attempt to mine FCCs that deliver "close" relationships among three dimensions. That is, we want to find the *maximum* patterns in a 3D context. The 3D pattern is maximum in that an increase in any dimension will cause a direct decrease in at least one of the other two dimensions; i.e., no further expansion in any dimension can be made on the pattern.

## 3. PRELIMINARIES

We shall first define some notations that we will be using throughout this paper, and then give the problem description.

Let $R = \{r_1, r_2, \ldots, r_n\}$ denote a set of rows, $C = \{c_1, c_2, \ldots, c_m\}$ denote a set of columns, and $H = \{h_1, h_2, \ldots, h_l\}$ denote a set of heights. Then a three-dimension dataset can be represented by a $l \times n \times m$ binary matrix $O = H \times R \times C = \{O_{k,i,j}\}$ with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$. Each cell $o_{kij}$ corresponds to the relationship among height $h_k$, row $r_i$, and column $c_j$. The value true (i.e., "1") denotes the relationship that any two dimensions are "simultaneously contained (S-contained)" in the third one.

Table 1 shows an example of a three-dimension dataset in Boolean context. In Table 1, $h_1$ and $r_4$ are S-contained in $c_3$ and $c_5$, denoted as $C(h_1 \times r_4) = \{c_3, c_5\}$; $h_2$ and $c_5$ are S-contained in $r_1$ and $r_4$, denoted as $R(h_2 \times c_5) = \{r_1, r_4\}$; $r_2$ and $c_1$ are S-contained in $h_1$ and $h_3$, denoted as $H(r_2 \times c_1) = \{h_1, h_3\}$.

**Table 1: Example of Binary Data Context.**

$H = h_1$

| $R/C$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $r_1$ | 1 | 1 | 1 | 0 | 1 |
| $r_2$ | 1 | 1 | 1 | 0 | 0 |
| $r_3$ | 1 | 1 | 1 | 1 | 1 |
| $r_4$ | 0 | 0 | 1 | 0 | 1 |

$H = h_2$

| $R/C$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $r_1$ | 1 | 1 | 1 | 1 | 1 |
| $r_2$ | 0 | 1 | 1 | 1 | 0 |
| $r_3$ | 1 | 1 | 1 | 1 | 0 |
| $r_4$ | 1 | 1 | 1 | 0 | 1 |

$H = h_3$

| $R/C$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|
| $r_1$ | 1 | 1 | 1 | 0 | 0 |
| $r_2$ | 1 | 1 | 1 | 0 | 0 |
| $r_3$ | 1 | 1 | 1 | 1 | 0 |
| $r_4$ | 1 | 1 | 0 | 1 | 1 |

**Definition 3.1 Height Support Set and H-Support:** *Given a set of rows $R' \subseteq R$ and a set of columns $C' \subseteq C$, the maximal set of heights that simultaneously contain $R'$ and $C'$ is defined as the Height Support Set $H(R' \times C') \subseteq H$. The number of heights in $H(R' \times C')$ is defined as the H-Support of $(R' \times C')$, denoted as $|H(R' \times C')|$.* For example, in Table 1, let $R' = \{r_1, r_2\}$ and $C' = \{c_1, c_2, c_3\}$, then $H(R' \times C') = \{h_1, h_3\}$ since both $h_1$ and $h_3$ simultaneously contain $\{r_1, r_2\}$ and $\{c_1, c_2, c_3\}$, and no other heights contain them simultaneously.

We can define **Row Support Set** $R(C' \times H')$ and **R-Support** $|R(C' \times H')|$, and also **Column Support Set** $C(R' \times H')$ and **C-Support** $|C(R' \times H')|$ in a similar way.

**Definition 3.2 Closed Cube:** *Given a set of rows $R' \subseteq R$, a set of columns $C' \subseteq C$, and a set of heights $H' \subseteq H$, a cube $A = (H' \times R' \times C') \subseteq O$ is defined as a Closed Cube if (1) $R' = R(C' \times H')$; (2) $C' = C(R' \times H')$; and (3) $H' = H(R' \times C')$.* For clarity, $A = (H' \times R' \times C')$ is written as $A = (H', R', C')$. Moreover, conditions (1), (2) and (3) are referred to as "closed" in row set, column set and height set respectively. Intuitively, a closed cube is complete (with all '1's inside) and maximal (no larger complete cubes contain it).

**Definition 3.3 Frequent Closed Cube (FCC):** *A cube $A = (H', R', C') \subseteq O$ is called a frequent closed cube if (1) the H-Support $|H(R' \times C')|$, R-Support $|R(H' \times C')|$, and C-Support $|C(R' \times H')|$ are higher than the minimum H-Support threshold (minH), minimum R-Support threshold (minR), and minimum C-Support threshold (minC) respectively; and (2) A is a closed cube.* For example, given that $minH = minR = minC = 2$, the cube $A = \{h_1, h_3\} \times \{r_1, r_2, r_3\} \times \{c_1, c_2, c_3\}$ will be a frequent closed cube in Table 1. However, $A' = \{h_1, h_3\} \times \{r_2, r_3\} \times \{c_1, c_2, c_3\}$ is not a frequent closed cube in that $\{r_2, r_3\} \neq R(\{h_1, h_3\} \times \{c_1, c_2, c_3\}) = \{r_1, r_2, r_3\}$. For clarity, cube $A' = \{h_1, h_3\} \times \{r_2, r_3\} \times \{c_1, c_2, c_3\}$ is written as $A' = (h_1h_3, r_2r_3, c_1c_2c_3)$.

**Problem Definition:** Given a three-dimension dataset $O$, our problem is to discover all frequent closed cubes with respect to the user support thresholds *minH*, *minR*, and *minC*.

# 4. REPRESENTATIVE SLICE MINING

In this section, we propose a framework, called Representative Slice Mining (RSM), to mine FCCs. Under this framework, any 2D FCP mining algorithms can be adapted to work on the 3D dataset. This framework is based on the idea that the 3D dataset $O = H \times R \times C$ can be presented as $O = H \times Slice_{R \times C}$. Hence, any dimension such as $H$ set can be enumerated first, which results in all possible combinations of slices. Then on each combination of slices, 2D FCP algorithms can be applied on the other two dimensions such as $R$ and $C$. Finally, a post-processing strategy is applied on the results to remove unclosed cubes due to the first enumerated dimension $H$. Based on this idea, we divide the RSM framework into three phases. In phase 1, representative slice is generated based on one-dimension enumeration and slices combination; in phase 2, any 2D frequent closed pattern mining algorithm can be applied to mine 2D FCPs on each representative slice; in phase 3, a post-pruning scheme is applied to remove FCCs unclosed in the enumerated dimension. We shall present the details of the three phases below, before discussing the correctness of the scheme.

## 4.1 Representative Slice Generation

In phase 1, we first take the height dimension $H$ as our *base* dimension[1], and enumerate set $H = \{h_1, h_2, \ldots, h_l\}$ to get all subsets of $H$ (denoted $H'$) such that $|H'| >= minH$. Given the dataset in Table 1 for example, let $minH = 2$, we will get the subsets $\{h_1, h_2\}$, $\{h_1, h_2, h_3\}$, $\{h_1, h_3\}$, and $\{h_2, h_3\}$.

Second, slices within the same subset are combined to form a new representative slice (RS). Given a 3D dataset $O = H \times R \times C = \{O_{k,i,j}\}$ with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$, and let $H' = \{h_1, \ldots, h_x\}$ be the subset to be combined. Then the RS of $H'$ can be represented as a $n \times m$ matrix such that $\forall O'_{i,j} \in RS, O'_{i,j} = \sum_{k=1}^{x} \cap O_{k,i,j}$ where $i \in [1, n]$ and $j \in [1, m]$. That is, the cell value of the representative slice is 1 only when all of its make-up values are 1; otherwise, the cell value is 0. And we say that the heights in $H'$ "contribute to" the RS of $H'$. The 2nd column of Table 2 shows the representative slices of the above example.

## 4.2 2D FCP Generation

In phase 2, any existing FCP mining algorithm can be applied on each representative slice to mine 2D FCPs based on dimensions $R$ and $C$. In our experiments, we adopted D-Miner [3] as it has been shown to be efficient on relatively dense datasets with long patterns. After mining, we will have a set of 2D FCPs for $R$ and $C$ dimensions. For our running example, the FCPs are shown in the 3rd column of Table 2.

## 4.3 3D FCC Generation by Post-pruning

In phase 3, 3D frequent patterns are generated by combining each 2D FCP with the heights contributing to its representative slice. However, not all those 3D frequent patterns are FCCs. Some of them are not closed in the height set and should be pruned off. For example, in Table 2, after combining the first 2D FCP "$r_1r_3 : c_1c_2c_3, 2 : 3$" with the contributing heights "$h_2, h_3$", a 3D frequent pattern "$h_2h_3 : r_1r_3 : c_1c_2c_3, 2 : 2 : 3$" is generated. This 3D frequent pattern is not a FCC in that it is unclosed in the height set and has a superset "$h_1h_2h_3 : r_1r_3 : c_1c_2c_3, 3 : 2 : 3$"(the 4th FCC in the 4th Column of Table 2). That is, the 2D FCP is not only contained in slices $h_2$ and $h_3$, but also contained in slice $h_1$.

To remove all unclosed 3D frequent closed patterns, we develop a post-pruning strategy based on Lemma 1. If a 2D FCP is contained in other height slices besides its contributing height slices, it is unclosed and hence can be removed; otherwise, it is retained.

LEMMA 1. *Post-pruning Strategy: Let $O' = H' \times R' \times C'$ be a 3D frequent pattern and $H$ be the whole height set. If $\exists H'' \in (H \setminus H')$ such that $\forall h_k \in H'', \forall r_i \in R', \forall c_j \in C', O_{k,i,j} = 1$, $O'$ is unclosed in the height set and can be pruned off; otherwise, $O'$ is retained.*

**Proof:** $\exists H'' \in (H \setminus H')$ such that $\forall h_k \in H'', \forall r_i \in R', \forall c_j \in C', O_{k,i,j} = 1$. So, there exists $O_s = ((H'' \cup H') \times R' \times C')$, which is the superset of $O' = (H' \times R' \times C')$. Hence, $O'$ is not closed in the height set, which contradicts the condition (3) of Closed Cube definition. That is, $O'$ is not a closed cube and should be pruned off. □

In the post pruning process, not all relative cells in all non-contributing slices are checked. During the slice checking process, any one cell with value '0' can stop one slice checking. And any slice passing the checking process (all relative cells value '1') without early termination can stop other slices' checking process in that the pattern is already

[1]Note that we can pick any of the dimensions as the base dimension. In fact, as we shall see, because the base dimension has to be enumerated over all combinations of its values, picking the dimension that has the smallest number of values is a good heuristic. WLOG, we shall use the height dimension for our discussion.

**Table 2: RSM Example ($minH = minR = minC = 2$).**

| Height Set | Representative Slices | 2D FCPs | 3D FCCs |
|---|---|---|---|
| $h_2, h_3$ | 11100<br>01100<br>11110<br>11001 | $r_1r_3 : c_1c_2c_3, 2:3$<br>$r_1r_3r_4 : c_1c_2, 3:2$<br>$r_1r_2r_3 : c_2c_3, 3:2$ | $h_2h_3 : r_1r_3r_4 : c_1c_2, 2:3:2$ |
| $h_1, h_3$ | 11100<br>11100<br>11110<br>00001 | $r_1r_2r_3 : c_1c_2c_3, 3:3$ | $h_1h_3 : r_1r_2r_3 : c_1c_2c_3, 2:3:3$ |
| $h_1, h_2$ | 11101<br>01100<br>11110<br>00101 | $r_1r_4 : c_3c_5, 2:2$<br>$r_1r_3 : c_1c_2c_3, 2:3$<br>$r_1r_2r_3 : c_2c_3, 3:2$ | $h_1h_2 : r_1r_4 : c_3c_5, 2:2:2$ |
| $h_1, h_2, h_3$ | 11100<br>01100<br>11110<br>00001 | $r_1r_3 : c_1c_2c_3, 2:3$<br>$r_1r_2r_3 : c_2c_3, 3:2$ | $h_1h_2h_3 : r_1r_3 : c_1c_2c_3, 3:2:3$<br>$h_1h_2h_3 : r_1r_2r_3 : c_2c_3, 3:3:2$ |

confirmed to be unclosed. This strategy ensures that we finish the close checking process as early as possible. For the example in Table 2, after the post-pruning process, the resulting FCCs are shown in the 4th column.

THEOREM 1. *Let $FCCs$ be the set of frequent closed cubes of a 3D dataset. Let $\xi$ denote the resultant frequent closed cubes obtained from running RSM on the dataset. Then $FCCs = \xi$. In other words, RSM correctly generates all and only all FCCs.*

**Proof:** Let $MineFCP(RS)$ denote the 2D FCP mining algorithm on slice $RS$. First, we prove that $FCCs \subseteq \xi$. Let $\delta$ be the set of unclosed 3D frequent patterns removed by the post-pruning strategy. Given any FCC $O' = H' \times R' \times C'$, then there must exist a representative slice $RS_{H'}$ such that $H'$ contributes to $RS_{H'}$. That is, $(R' \times C') \subseteq RS_{H'}$. Since $R' \times C'$ is closed for $H'$, $(R' \times C') \subseteq MineFCP(RS_{H'})$. Hence, $O' \in (\xi \cup \delta)$. As proved in Lemma 1, the post-pruning strategy only removes unclosed 3D frequent patterns, so $O' \notin \delta$. Thus, $O' \in \xi$. Hence, we conclude that $FCCs \subseteq \xi$.

Next, we prove $\xi \subseteq FCCs$ by contradiction. Assume there exists a 3D pattern $O' \in \xi$ but $O' \notin FCCs$. Then $O'$ is either not satisfied by monotonic support constraints or not closed. Suppose that $O' = H' \times R' \times C'$ does not satisfy $minH$ threshold, then $RS_{H'}$ will be pruned off during subset enumeration, and $O'$ will not be generated. Suppose that $O'$ does not satisfy $minR$ or $minC$ threshold, then $(R' \times C')$ of $O'$ will be pruned off during 2D FCP generation, and $O'$ will not be generated. This is contrary to the assumption. Hence, we gather that $O'$ satisfies monotonic support constraints but it is not closed.

Suppose that $O'$ is not closed in the $H$ set, then there exists a closed FCC $O'' = (H' \cup H_a) \times R' \times C'$ such that $\forall h_k \in H_a, r_i \in R', c_j \in C', O_{k,i,j} = 1$, where $H_a \in (H \setminus H')$. Hence, in the post-pruning process, $O'$ is pruned off, which is contrary to the assumption that $O' \in \xi$. Thus, we conclude that $O'$ is closed in the $H$ set.

Suppose that $O'$ is not closed in the $R$ set, then there exists a closed FCC $O'' = H' \times (R' \cup R_a) \times C'$ such that $\forall h_k \in H', r_i \in (R' \cup R_a), c_j \in C', O_{k,i,j} = 1$, where $R_a \subseteq (R \setminus R')$. Hence, $((R' \cup R_a) \times C') \subseteq RS_{H'}$. Then $((R' \cup R_a) \times C') \subseteq MineFCP(RS_{H'})$ and $R' \times C'$ is pruned off by the 2D FCP mining algorithm in that it is unclosed in the row set. Hence, $O'$ cannot be generated, which is contrary to the assumption that $O' \in \xi$. Thus, we conclude that $O'$ is closed in the $R$ set. Using the same logic, we can prove that $O'$ is closed in the $C$ set.

Now that we conclude that $O'$ is closed and satisfies all monotonic constraints. Hence, $O' \in FCCs$ and our assumption that there exists a 3D pattern $O' \in \xi$ but $O' \notin FCCs$ is wrong. That is, $\xi \subseteq FCCs$. So, our RSM framework for mining $FCCs$ is correct in that $\xi = FCCs$. □

# 5. CUBEMINER

While RSM has the advantage that it can reuse existing FCP mining algorithms, the number of 2D slices could be large. In this section, we present a novel approach that mine FCCs directly from the 3D dataset. We shall first present the principle behind our proposed CubeMiner scheme. Then, we will look at the algorithm, and finally we shall show the correctness of CubeMiner.

## 5.1 CubeMiner Principle

CubeMiner is a novel algorithm for mining FCC $(H', R', C')$ under constraints. It builds the sets $H'$, $R'$, and $C'$ and uses monotonic support threshold constraints simultaneously on $H$, $R$, and $C$ to reduce the search space. A FCC indicates that all its heights, rows, and columns are in "S-contained" relation.

We use $Z$ to denote a set of cell groups which are partitions of the false values (i.e., "0") of the boolean matrix. An element $(W, X, Y) \in Z$ is called a "cutter" if $\forall h_k \in W$, $\forall r_i \in X$, and $\forall c_j \in Y$, $O_{k,i,j} = 0$. And we call $W, X, Y$ the left atom, middle atom, and right atom of cutter $(W, X, Y)$ respectively. We summarize the "0" cells row by row, hence, $Z$ contains as many cutters as rows in all height slices of the 3D data matrix. Each cutter is composed of the cell valued by 0 in the row. Table 3 shows the 10 cutters of the matrix in Table 1. The cutters are sorted by ascending order of left atom first and middle atom second.

CubeMiner starts with the whole dataset $O(H, R, C)$ and then splits it recursively using the cutters of $Z$ until all cutters in $Z$ are used and consequently all cells in each resulting

**Table 3: $Z$(cutter set).**

| $W, X, Y$ |
|---|
| $h_1, r_1, c_4$ |
| $h_1, r_2, c_4c_5$ |
| $h_1, r_4, c_1c_2c_4$ |
| $h_2, r_2, c_1c_5$ |
| $h_2, r_3, c_5$ |
| $h_2, r_4, c_4$ |
| $h_3, r_1, c_4c_5$ |
| $h_3, r_2, c_4c_5$ |
| $h_3, r_3, c_5$ |
| $h_3, r_4, c_3$ |

cube have the value 1. A cutter $(W, X, Y)$ in $Z$ is used to cut a cube $(H', R', C')$ if $W \cap H' \neq \emptyset$, $X \cap R' \neq \emptyset$, and $Y \cap C' \neq \emptyset$. In this case, we say that the cutter is "applicable" to the cube. By convention, we define the left son of $(H', R', C')$ by $(H' \setminus W, R', C')$, the middle son by $(H', R' \setminus X, C')$ and the right son by $(H', R', C' \setminus Y)$. Recursive splitting leads to all FCCs, but also some non-maximal unclosed cubes. Pruning Strategies need to be applied to ensure that we obtain all FCCs and only the FCCs. We shall consider how to develop such pruning strategies. Figure 1 shows the tree generated from the 3D matrix in Table 1.

From Figure 1, we see that the 10 cutters in Table 3 split the original dataset into the resulting leaves in 10 steps (levels). We define the steps from the root to a node as the node's "path". Each node is split into three new nodes in the next level if the cutter is applicable. We only keep and show nodes satisfying support thresholds (given $minH = minR = minC = 2$) due to space limitation. However, in each level, not all nodes generated are useful for further splitting. There are four categories of useless nodes:

(a) Left son from a middle/right branch by the cutter whose left atom has cut the node's path before. For example, the left atom $h_1$ of cutter $(h_1, r_2, c_4c_5)$ has already cut the paths of left sons $L(h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ ($a_1$ in Level 2) and $L(h_2h_3, r_1r_2r_3r_4, c_1c_2c_3c_5)$ ($a_2$ in Level 2) in Level 2. $a_1$ from the middle branch is unclosed in row set and $a_2$ from the right branch is unclosed in column set. They are to be pruned off as the subsets of node $L(h_2h_3, r_1r_2r_3r_4, c_1c_2c_3c_4c_5)$ (1st node in Level 1).

(b) Middle son from a right branch by the cutter whose middle atom has cut the node's path before. For example, the middle atom $r_2$ of cutter $(h_2, r_2, c_1c_5)$ has already cut the path of middle son $M(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ ($b_1$ in Level 4). This middle son is unclosed in column set and should be pruned off as the subset of node $M(h_1h_2h_3, r_1r_3, c_1c_2c_3c_5)$ (2nd node in Level 3). Middle sons $b_2$, $b_3$ and $b_4$ are all in such cases: they are either duplicates or subsets of other nodes.

(c) Nodes that are unclosed in height set. For example, node $R(h_2h_3, r_1r_3, c_1c_2c_3)$ ($c_1$ in Level 7) is unclosed in height set because there exists its superset node $R(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ (5th node in Level 5). Such nodes should be pruned off to ensure closure in height set. Nodes $c_2, c_3, c_4$ are all such examples.

(d) Nodes that are unclosed in row set. For example, node $R(h_1h_2h_3, r_2r_3r_4, c_1c_2c_3)$ ($d_1$ in Level 2) is unclosed in row set because there exists its superset node $R(h_1h_2h_3, r_1r_2r_3r_4, c_1c_2c_3)$ (6th node in Level 2). Such nodes should be pruned off to ensure closure in row set. Node $R(h_2h_3, r_1r_4, c_1c_2c_3)$ ($d_2$ in Level 7) is also one such example to be pruned off as it is not closed due to row $r_3$. Note that there exists some nodes that are closed in row set although they may have a temporary superset node in the processing. For example, node $R(h_1h_2h_3, r_3r_4, c_3c_5)$ ($d_3$ in Level 3) has a temporary superset node $R(h_1h_2h_3, r_1r_3r_4, c_3c_5)$ ($d_4$ in Level 3). Though node $d_3$ appears to be temporarily 'unclosed' due to row $r_1$, we detect that after applying a later cutter $(h_3, r_1, c_4c_5)$ in level 7, node $d_4$ loses its superset status, and $d_3$'s offspring $L(h_1h_3, r_3r_4, c_3c_5)$ ($d_5$ in Level 7) just serves as a reason to remove the middle son $M(h_1h_3, r_3r_4, c_3c_5)$ ($b_2$, an offspring of $d_4$) safely. Hence, such row set nodes which are temporary unclosed during processing are retained in that they are row set closed in the whole scenario.

To remove useless nodes of (a) and (b) types, we maintain two sets $TL = \{W_1, W_2, \ldots, W_p\}, TM = \{X_1, X_2, \ldots, X_q\}$ in each node to keep track of the left and middle atoms of cutters that cut its path. And based on the two sets, we develop Left Track Checking in Lemma 2 and Middle Track Checking in Lemma 3. In the initial status, $TL = TM = \emptyset$ for the root. Since only left sons from a middle/right branch need to be checked, $TL$ set is updated only on a newly generated middle/right son. Similarly, since only middle sons from a right branch need to be checked, $TM$ set is updated only on a newly generated right son. We shall denote the $TL$ (and $TM$) set of node $O$ as $TL_O$ (and $TM_O$).

LEMMA 2. *Left Track Checking: Let $L = (H' \setminus W, R', C')$ be the left son of node $O' = (H', R', C')$ by cutter $z = (W, X, Y)$. If $W \cap TL_{O'} \neq \emptyset$, $L$ can be pruned off.*

**Proof:** Since $W \cap TL_{O'} \neq \emptyset$, $W \subseteq TL_{O'}$, hence $\exists z' = (W, X', Y') \in Z$ cuts $O'$'s ancestor $O'_a = (H'_a, R'_a, C'_a)$. Let $O_l = (H_l, R_l, C_l)$ be the left sibling of $O'_a$ by cutter $z'$. Then, either (1) $H_l = H'_a \setminus W, R'_a = R_l \setminus X' \subset R_l, C'_a = C_l$ or (2) $H_l = H'_a \setminus W, R'_a = R_l, C'_a = C_l \setminus Y' \subset C_l$. Since cutters between $z'$ and $z$ are all with left item $W$, which are not applicable to $O_l$, $O_l$ remains unchanged after all $z'$ to $z$ cuttings and $H' \setminus W = H'_a \setminus W = H_l, R' \subseteq R_a, C' \subseteq C'_a$. So, in both (1) and (2), we can draw the conclusion that $L \subset O_l$. Hence, $L$ can be pruned off. □

For example, in Figure 1, the left son $L(h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ (a1 in level 2) of parent $P(h_1h_2h_3, r_2r_3r_4, c_1c_2c_3c_4c_5)$ (2nd node in level 1) by cutter $(h_1, r_2, c_4c_5)$ is pruned off in that $W \cap TL_p = \{h_1\} \neq \emptyset$.

LEMMA 3. *Middle Track Checking: Let $M = (H', R' \setminus X, C')$ be the middle son of node $O' = (H', R', C')$ by cutter $z = (W, X, Y)$. If $X \cap TM_{O'} \neq \emptyset$, $M$ can be pruned off.*

We can use the same logic in Lemma 2 to prove the correctness of Lemma 3, and hence we omit it here.

For example, in Figure 1, the middle son $M(h_1h_2h_3, r_1r_3, c_1c_2c_3)$ (b1 in level 4) of parent $P(h_1h_2h_3, r_1r_2r_3, c_1c_2c_3)$ (4th node in level 3) by cutter $(h_2, r_2, c_1c_5)$ is pruned off in that $X \cap TM_p = \{r_2\} \neq \emptyset$.

To remove useless nodes of (c) and (d) types, we develop Close Height Set Checking in Lemma 4 and Close Row Set Checking in Lemma 5

LEMMA 4. *Close Height Set Checking: Let $O'' = (H'', R'', C'')$ be the middle/right son of node $O'$ and $Z$ be the whole cutter set. If $\exists h_w \in (H \setminus H'')$ ($H$ is the full height set*

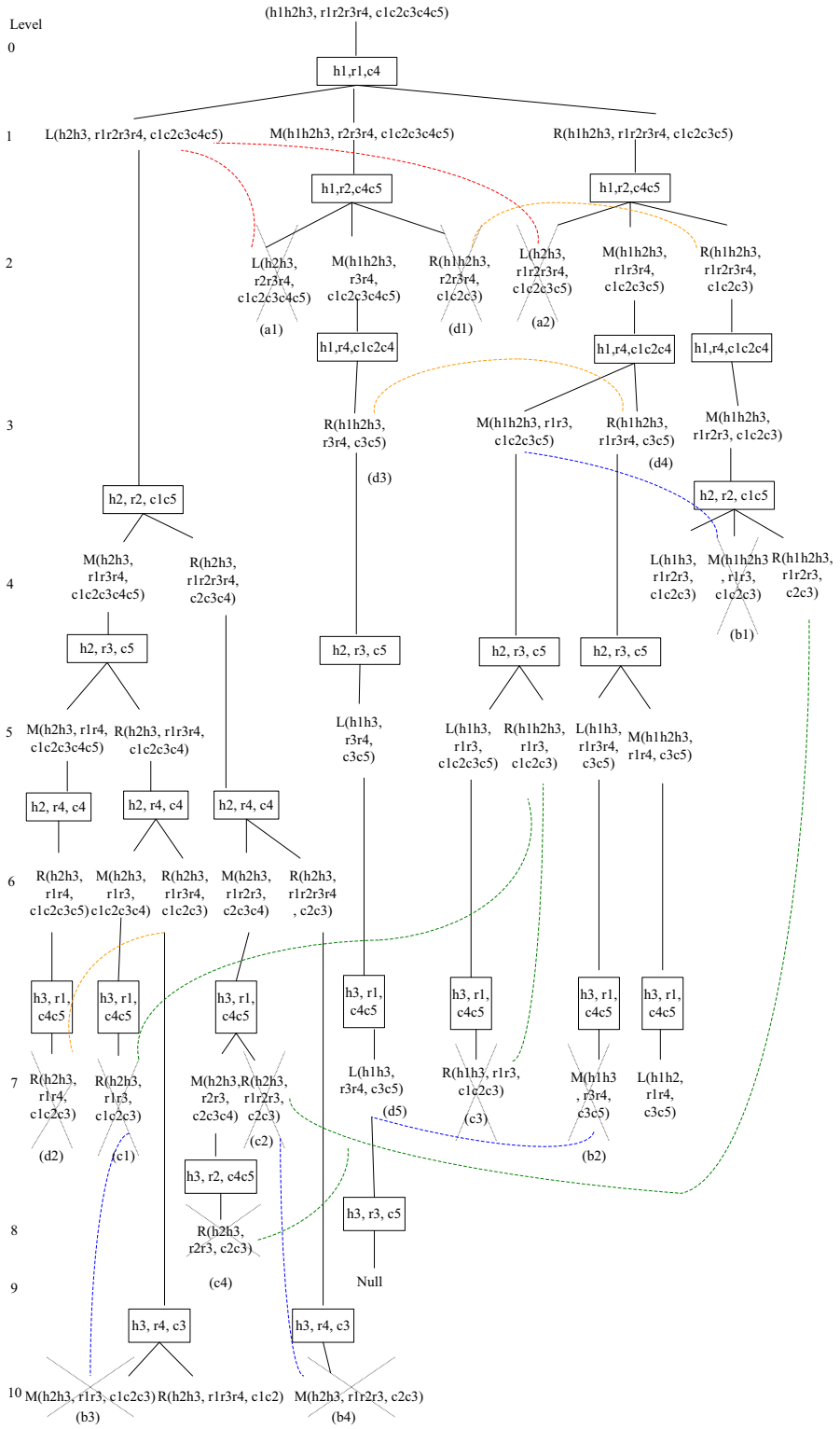



**Figure 1: FCC Mining Tree.**

*of $O$) such that $\forall(\{h_w\},\{r_x\},C_y) \in Z$ where $r_x \in R''$, $C'' \cap C_y = \emptyset$, then $O''$ is unclosed in the height set and can be pruned off. Since the left son never satisfies the condition, only the middle and right sons need this checking.*

**Proof:** $\exists h_w \in (H \setminus H'')$ such that $\forall(\{h_w\},\{r_x\},C_y) \in Z$ where $r_x \in R'', C'' \cap C_y = \emptyset$, i. e., $\forall O_{w,x,y} \in (\{h_w\}, R'', C'')$, $O_{w,x,y} = 1$. So, there exists $O_s = (H'' \cup \{h_w\}, R'', C'')$, which is the superset of $O'' = (H'', R'', C'')$. Hence, $O''$ is not closed in the height set and can be pruned off. □

For example, in Figure 1, node $R(h_2h_3, r_1r_2r_3, c_2c_3)$ (c2 in level 7) is not closed in the height set because there is $h_1 \in (H \setminus \{h_2, h_3\})$ such that for cutters $(h_1, r_1, c_4)$ and $(h_1, r_2, c_4c_5)$, $\{c_2, c_3\} \cap \{c_4\} = \emptyset$ and $\{c_2, c_3\} \cap \{c_4c_5\} = \emptyset$. And we find c2's superset in node $R(h_1h_2h_3, r_1r_2r_3, c_2c_3)$ (5th node in level 4).

LEMMA 5. *Close Row Set Checking: Let $O'' = (H'', R'', C'')$ be the left/right son of node $O'$ and $Z$ be the whole cutter set. If $\exists r_x \in (R \setminus R'')$ ($R$ is the full row set of $O$) such that $\forall(\{h_w\},\{r_x\},C_y) \in Z$ where $h_w \in H''$, $C'' \cap C_y = \emptyset$, then $O''$ is unclosed in the row set and can be pruned off. Since the middle son never satisfies the condition, only the left and right sons need this checking.*

Again, we can use the same logic in Lemma 4 to prove the correctness of Lemma 5, so it is omitted here.

For example, in Figure 1, node $R(h_2h_3, r_1r_4, c_1c_2c_3)$ (d2 in level 7) is not closed in the row set because $\exists r_3 \in (R \setminus \{r_1, r_4\})$ such that for cutters $(h_2, r_3, c_5)$ and $(h_3, r_3, c_5)$, $\{c_1, c_2, c_3\} \cap \{c_5\} = \emptyset$. And we find d2's superset in node $R(h_2h_3, r_1r_3r_4, c_1c_2c_3)$ (3th node in level 6).

## 5.2 Algorithm CubeMiner

We are now ready to present CubeMiner algorithmically. CubeMiner is a depth-first method to mine FCCs. Algorithm 1 contains the pseudo-code of CubeMiner. First, the left/middle track set $TL/TM$ is initialized with empty set and the set $Z$ of cutters is computed, and then the recursive function *cut*() in Algorithm 2 is called.

**Algorithm 1** CubeMiner

```
1: CubeMiner()
2: Global variables: H the set of heights, R the set of rows,
   C the set of columns, monotonic constraints minH,
   minR, and minC on H, R, C respectively.
3: Input: 3D Matrix O with l heights, n rows and m
   columns.
4: Output: ξ the set of FCCs.
5: TL ← empty(), TM ← empty();
6: Z and |Z| are computed from O;
7: ξ ← cut((H, R, C), Z, 0, |Z|, TL, TM);
```

Function *cut*() cuts a node $O' = (H', R', C')$ with the first cutter $Z[i] = (W, X, Y)$ that satisfies the following constraints. First, $(H', R', C')$ must have a non empty intersection with $Z[i]$. If it is not the case, *cut*() is called with the next cutter.

To build the left son $L = (H' \setminus W, R', C')$ (lines 9-14), three checks are required: monotonic constraint check $minH(H' \setminus W)$, left track check, and close row set check (*Rcheck*() in Algorithm 4). If $L$ is not pruned off by the three checks, *cut*() is called to process $L$, and there is no update on $TL, TM$ sets for $L$.

To build the middle son $M = (H', R' \setminus X, C')$ (lines 15-20), three checks are required: monotonic constraint check $minR(R' \setminus X)$, middle track check, and close height set check (*Hcheck*() in Algorithm 3). If $M$ is not pruned off by the three checks, *cut*() is called to process $M$, and the $TL$ set for $L$ is updated to $TL \cup W$.

To build the right son $R = (H', R', C' \setminus Y)$ (lines 21-29), three checks are required: monotonic constraint check $minC(C' \setminus Y)$, close height set check and close row set check. If $R$ is not pruned off by the three checks, *cut*() is called to process $R$, and the $TL, TM$ sets for $L$ are updated to $TL \cup W, TM \cup X$ respectively.

Since the size of $Z$ and the order of cutters inside $Z$ are important to performance, the algorithm can be optimized by preprocessing the 3D dataset. We adopt two heuristics. First, we transpose the 3D data matrix to make $|H| < |C|$ and $|R| < |C|$, which helps to minimize the size of $|Z|$. Second, we sort the height slices such that height slices with more 0s are always in front of those with fewer 0s. This helps to accelerate the mining by pruning off the search space as early as possible.

**Algorithm 2** Cutting

```
1: cut((H', R', C'), Z, 0, |Z|, TL, TM)
2: Input: Node (H', R', C'), cutters list Z, iteration num-
   ber i, |Z| the size of Z, left and right atoms tracks TL
   and TM.
3: Output: ξ the set of FCCs.
4: (W, X, Y) ← Z[i];
5: if i ≤ |Z| − 1 then
6:   if W ∩ H' = ∅ or X ∩ R' = ∅ or Y ∩ C' = ∅ then
7:     ξ ← ξ ∪ cut((H', R', C'), Z, i + 1, |Z|, TL, TM);
8:   else
9:     if minH(H' \ W) satisfied and W ∩ TL = ∅ then
10:      β ← Rcheck((H' \ W, R', C'), Z);
11:      if β = 1 then
12:        ξ ← ξ ∪ cut((H' \ W, R', C'), Z, i + 1, |Z|, TL, TM);
13:      end if
14:    end if
15:    if minR(R' \ X) satisfied and X ∩ TM = ∅ then
16:      α ← Hcheck((H', R' \ X, C'), Z);
17:      if α = 1 then
18:        ξ ← ξ ∪ cut((H', R' \ X, C'), Z, i + 1, |Z|, TL ∪ W, TM);
19:      end if
20:    end if
21:    if minC(C' \ Y) satisfied then
22:      α ← Hcheck((H', R', C' \ Y), Z);
23:      if α = 1 then
24:        β ← Rcheck((H', R', C' \ Y), Z);
25:        if β = 1 then
26:          ξ ← ξ ∪ cut((H', R', C' \ Y), Z, i + 1, |Z|, TL ∪ W, TM ∪ X);
27:        end if
28:      end if
29:    end if
30:  end if
31: else
32:   ξ ← (H', R', C');
33: end if
34: return ξ;
```

**Algorithm 3** Close Height Set Check

1: **Hcheck**$((H', R', C'), Z)$
2: Input: node $(H', R', C')$ and cutters list $Z$.
3: Output: flag $\alpha$.
4: **if** $\exists h_w \in (H \backslash H')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $r_x \in R'$ , $C' \cap C_y = \emptyset$ **then**
5: $\alpha \leftarrow 0$;
6: **else**
7: $\alpha \leftarrow 1$;
8: **end if**
9: **return** $\alpha$;

**Algorithm 4** Close Row Set Check

1: **Rcheck**$((H', R', C'), Z)$
2: Input: node $(H', R', C')$ and cutters list $Z$.
3: Output: flag $\beta$.
4: **if** $\exists r_x \in (R \backslash R')$ such that $\forall(\{h_w\}, \{r_x\}, C_y) \in Z$ where $h_w \in H'$, $C' \cap C_y = \emptyset$ **then**
5: $\beta \leftarrow 0$;
6: **else**
7: $\beta \leftarrow 1$;
8: **end if**
9: **return** $\beta$;

THEOREM 2. *Let $FCCs$ be the set of frequent closed cubes of a 3D dataset. Let $LV$ be the set of leaf nodes derived from CubeMiner on the dataset. Then, $FCCs = LV$. In other words, CubeMiner can correctly generate all and only all FCCs.*

**Proof:** First, we prove that $FCCs \subseteq LV$. Let $(H, R, C)$ be the original dataset, $Z$ be the whole cutter set and $P$ be the set of pruned nodes. Since $FCCs \subseteq (H, R, C)$, and in the tree building process, only cells valued '0' are removed off by cutters (verified by node's son definition) and only useless nodes (subsets of other nodes) are pruned off (verified by Lemma 2 to Lemma 5), hence, $FCCs \subseteq (H, R, C) \setminus Z \setminus P$, that is, $FCCs \subseteq LV$.

Second, we prove $LV \subseteq FCCs$ by contradiction. Assume there exists a leave $A \in LV$ but $A \notin FCCs$. Then $A$ is either not satisfied by monotonic support constraints or not closed. Let $A = (H_a, R_a, C_a)$ and $Z_l, Z_m, Z_r$ be the set of cutters associated to the left, middle, right branches of the path from the root to $A$ respectively. During the tree building process, each time we cut off a node's height set, the monotonic constraint $minH$ is checked to be satisfied, hence, $H_a$ satisfies the monotonic support constraint. Similarly, $R_a$ and $C_a$ both satisfy their monotonic constraints. Hence, we gather that $A$ is not closed.

Suppose that $A$ is not closed in the column set, then there exists $A' = (H_a, R_a, C_a \cup C'_a)$ where $C'_a \subseteq C \setminus C_a$, and $\forall h_k \in H_a, \forall r_i \in R_a, \forall c_j \in C_a \cup C'_a, O_{k,i,j} = 1$. And since the whole column set $C$ is cut into $C_a$ from the root to $A$ by cutters in $Z_r$, so there exists a set of cutters $Z_a \subseteq Z_r$ to cut off $C'_a$ and $\forall(W, X, Y) \in Z_a$, either (a) $W \subseteq H \setminus H_a$ or (b) $X \subseteq R \backslash R_a$. Given any of $A$'s ancestor $B = (H_b, R_b, C_b)$ derived from a cutter $(W, X, Y) \in Z_a$. Since $B$ is a right son, $W \subseteq H_b, X \subseteq R_b$, and the $TL$ and $TM$ sets are updated into $TL \cup W$ and $TM \cup X$ respectively. For case (a), $W \nsubseteq H_a$, there must exist a cutter in $Z_l$ to remove off $W$ on the path from $B$ to $A$. That is, between $B$ and $A$, there must exist a left-son offspring of $B$. However, since the left atom of the cutter $W_l \cap TL = W \neq \emptyset$, the left-son offspring is pruned off and hence no $A$ will be generated, which is contrary to the previous assumption. For case (b), it is similar to (a): during the process to remove off $X$ from $R_b$, the middle-son offspring of $B$ is pruned off due to $X_m \cap TM = X \neq \emptyset$. As a result, $A$ will not be generated and it is contrary to the assumption too. Hence, we conclude that the assumption is wrong and $A$ is closed in the column set.

Suppose that $A$ is not closed in the row set, then there exists $A' = (H_a, R_a \cup R'_a, C_a)$ where $R'_a \subseteq R \setminus R_a$, and $\forall h_k \in H_a, \forall r_i \in R_a \cup R'_a, \forall c_j \in C_a, O_{k,i,j} = 1$. And since the whole column set $R$ is cut into $R_a$ from root to $A$ by cutters in $Z_m$, so there exists a set of cutters $Z_a \subseteq Z_m$ to cut off $R'_a$ and $\forall(W, X, Y) \in Z_a$, either (c) $W \subseteq H \setminus H_a$ or (d) $Y \subseteq C \setminus C_a$. Given any of $A$'s ancestor $B = (H_b, R_b, C_b)$ obtained from a cutter $(W, X, Y) \in Z_a$. Since $B$ is a middle son, $W \subseteq H_b, Y \subseteq C_b$, and the $TL$ set is updated into $TL \cup W$. Case (c)'s proof is the same as case (a) above. As for case (d), $Y \nsubseteq C_a$, there must exist cutters in $Z_r$ to remove off $Y$ on the path from $B$ to $A$. Let $B' = (H_b, R_b, C_a)$ be the right-son offspring of $B$ after removing $Y$. Since $X \cap R_b = \emptyset$, and $X \cap R'_a \neq \emptyset$, $\exists r_u = X \cap R'_a$ such that $\forall h_k \in H_a, \forall c_j \in C_a, O_{k,u,j} = 1$. Hence $B'$ is not row set closed due to $r_u$ and will be pruned off in the close row set checking of right son building process. As a result, $A$ will not be generated, which is contrary to the assumption. Hence we conclude that the assumption is wrong and $A$ is closed in the row set.

Suppose that $A$ is not closed in the height set, then there exists $A' = (H_a \cup H'_a, R_a, C_a)$ where $H'_a \subseteq H \setminus H_a$, and $\forall h_k \in H_a \cup H'_a, \forall r_i \in R_a, \forall c_j \in C_a, O_{k,i,j} = 1$. And since the whole height set $H$ is cut into $H_a$ from the root to $A$ by cutters in $Z_l$, so there exists a set of cutters $Z_a \subseteq Z_l$ to cut off $H'_a$ and $\forall(W, X, Y) \in Z_a$, either (e) $X \subseteq R \setminus R_a$ or (f) $Y \subseteq C \setminus C_a$. Like the proof in case (d), in case (e)/(f), the ancestor of $A$ will be pruned off as it will be unclosed in the height set checking during middle/right son building process. Hence, $A$ will not be generated, and it is contrary to the assumption. We conclude that the assumption is wrong and $A$ is closed in the height set.

Now, we have concluded that $A$ is closed and satisfies all monotonic constraints. Hence, $A \in FCCs$ and our assumption that there exists a leave $A \in LV$ but $A \notin FCCs$ is wrong. That is, $LV \subseteq FCCs$. So, our algorithm for mining $FCCs$ is correct in that $LV = FCCs$. □

# 6. PARALLEL FCC MINING

Given that FCC mining is computationally expensive, a solution to reduce the response time is to exploit parallelism. In this section, we shall show how our proposed RSM and CubeMiner can be parallelized easily.

In general, a parallel algorithm typically comprises three logical phases: (a) a task generation phase that splits the original task into smaller sub-tasks; (b) a task allocation phase that assigns the sub-tasks to the processors; (c) a task execution phase where every processor operates on the allocated sub-tasks. An important factor in parallelism is to minimize interference during the execution phase, so that all processors can operate independently and concurrently without having to communicate with one another.

It turns out that both RSM and CubeMiner fit nicely into

the above framework: tasks can be generated and allocated to processors to be executed independently.

- **Parallel RSM.** In RSM, the mining of each representative slice corresponds to a task, in other words, the maximum number of tasks is the number of eumerations of the base dimensions (those enumerations that do not meet the minimum thresold requirement are dropped). Each of these tasks can be allocated to a processor, and can be processed independently.

- **Parallel CubeMiner.** In CubeMiner, each branch of the tree splitting process can be processed independently, and thus, each branch corresponds to a task. In other words, we can allocate a branch of the tree splitting process to a processor.

For both Parallel-RSM and Parallel-CubeMiner, to ensure that the tasks can be processed independently each processor requires a copy of the entire dataset. This is necessary so that the post-pruning phase can be performed independently. Fortunately, the communication overhead (to transmit the dataset to all processors) is not significant: (a) the dataset can be transmitted while the tasks are being generated, so the response time is not much affected; (b) the communication cost is relatively small compared to the mining cost.

# 7. EXPERIMENTAL RESULTS

We have implemented the RSM framework and CubeMiner in C. For the RSM framework, we employed D-Miner [3] in phase two as the 2D FCP mining scheme. This is because D-Miner keeps the supporting row set of each FCP during the processing, which is important for close check of 3D FCC. Moreover, D-Miner has been shown to perform well in relatively dense datasets. We conducted a performance study to evaluate the efficiency of RSM against CubeMiner, and study the optimization of CubeMiner. In addition, we also study the parallel versions of RSM and CubeMiner. For our experiments, we use two real 3D microarray datasets: the yeast cell-cycle regulated genes [15] (http://genomewww.stanford.edu/cellcycle) in the Elutriation Experiments and CDC15 Experiments respectively. To study the effect of the proposed schemes on scalability, we also use synthetic datasets generated by the IBM data generator. The generator is available at http://www.cs.umbc.edu /∼cgiannel/assoc_gen.html. All the experiments are run on a Pentium 4 PC with 1 GB RAM.

## 7.1 Results from Real Microarray Datasets

In this section, we experiment on two real microarray datasets. For the Elutritration Experiments, there are a total of 7161 genes whose expression values are measured from time 0 to 390 minutes at 30 minute intervals (a total of 14 time points). And for the CDC15 Experiments, there are a total of 7761 genes whose expression values are measured from time 70 to 250 minutes at 10 minute intervals (a total of 19 time points). Finally, we use 9 of the attributes of the raw data as the samples (e.g., the raw values for the average and normalized signal for Cy5 and Cy3 dyes, the ratio of those values, etc.) [19]. Thus, from the Elutritration dataset, we obtain a 3D expression matrix of size: $T \times S \times G = 14 \times 9 \times 7161$; and from the CDC15 dataset, we obtain a 3D expression matrix of size: $T \times S \times G = 19 \times 9 \times 7761$.

We normalize the 3D datasets to make its cell value '1' or '0', where value '1' means high expression value and '0' otherwise. For dataset $O' = T \times S \times G = \{O'_{k,i,j}\}$ with $k \in [1, l]$, $i \in [1, n]$ and $j \in [1, m]$. We normalize $O'$ into a $T \times S \times G$ matrix $O$ as follows:

$$O_{k,i,j} = \begin{cases} 1 & \text{if } O'_{k,i,j} \geq \frac{\sum_{j=1}^{m} O'_{k,i,j}}{m}, \\ 0 & \text{if } O'_{k,i,j} < \frac{\sum_{j=1}^{m} O'_{k,i,j}}{m}. \end{cases}$$

### 7.1.1 CubeMiner Optimization

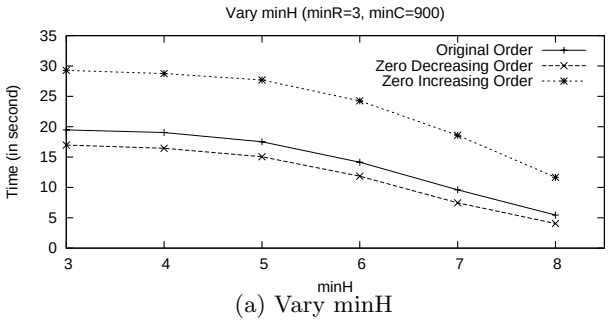
Before comparing the performance of CubeMiner and the RSM framework, we first study the optimization of CubeMiner. We experiment on the Elutritration dataset and sort the original dataset by Time Slice. We first sort the time slice such that time slices with more 0s are always in front of those with fewer 0s, which is called "Zero Decreasing Order"; then we sort the time slice such that time slices with fewer 0s are always in front of those with more 0s, which is called "Zero Increasing Order". We compare the performance of CubeMiner on the original order, Zero Decreasing Order and Zero Increasing Order. Figure 2 shows the results as we vary $minH$, $minR$ and $minC$ respectively. First, we observe that with the increase in $minH$, $minR$ and $minC$ values, regardless of the ordering of the datasets, the processing time of CubeMiner decreases. This is expected since a larger threshold value means that we can prune a larger space as the answer size is smaller. Second, in all the three cases, we observe that CubeMiner performs best on the dataset with Zero Decreasing Order, and worst on the dataset with Zero Increasing Order. The performance of CubeMiner on the original dataset stays in the middle position. CubeMiner performs best when the dataset is sorted by the Zero Decreasing Order because applying cutters with more 0s first will remove the patterns that do not satisfy the minimum thresholds early. That is, it helps to prune off the search space early, and hence accelerates the mining process. Based on these results, in the following experiments, we adopt an optimized version of CubeMiner that pre-sorts the datasets in Zero Decreasing Order before performing FCC mining.

### 7.1.2 Vary Monotonic Constraints

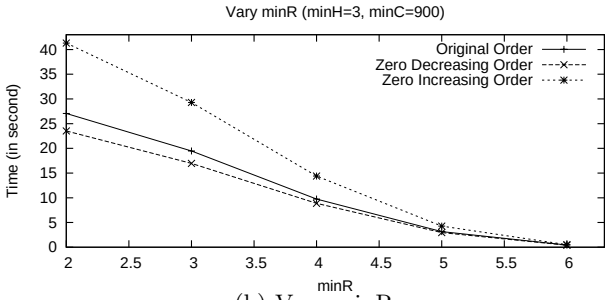
In this experiment, we vary the monotonic support constraints $minH$, $minR$ and $minC$, and study the performance of RSM and CubeMiner respectively. For RSM, we examine two versions using dimensions $H$ and $R$ as the base dimensions respectively. We denote these versions as 'RSM-H' and 'RSM-R' respectively. As we will be enumerating the $H$ and $R$ dimensions, the constraint $minC$ on dimension $C$ will have a relatively smaller effect. Hence, we study the effect of $minC$ first.

The results are shown in Figure 3. In Figure 3(a), we see clearly that RSM-R is much faster than RSM-H. This is because $|R| < |H|$ and a larger enumerated dimension leads to more representative slices. Hence, the enumeration on the smallest dimension always leads to better performance of RSM. When we refer to RSM in the following experiments, we default it as taking the smallest dimension to enumerate.

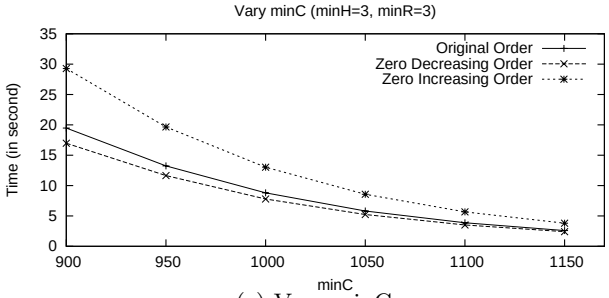
We also see that the execution time of CubeMiner and RSM-R both decrease with the increase of $minC$. Moreover, for the Elutritration dataset, RSM-R is faster than



(a) Vary minH

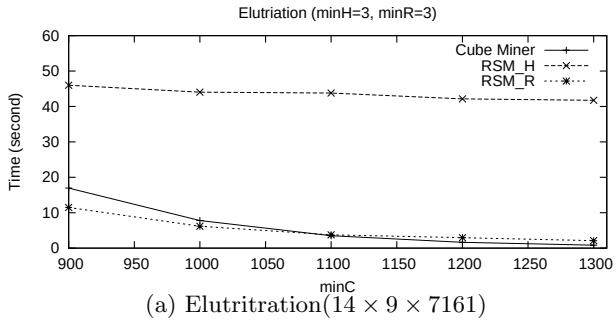


(b) Vary minR

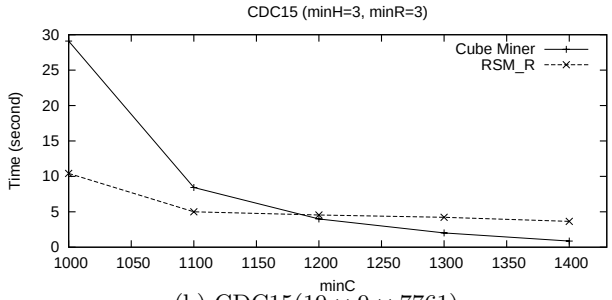


(c) Vary minC

**Figure 2: CubeMiner Optimization.**



(a) Elutritration($14 \times 9 \times 7161$)



(b) CDC15($19 \times 9 \times 7761$)

**Figure 3: Vary $minC$.**

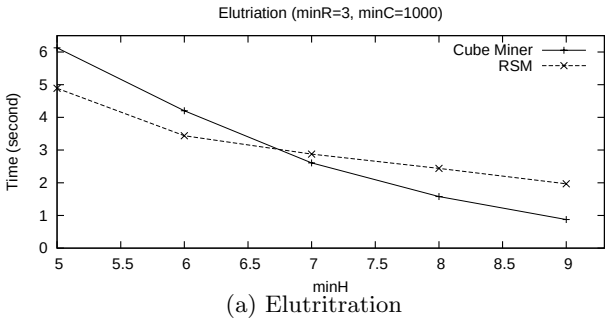
CubeMiner when $minC$ is below 1000. However, CubeMiner catches up and performs better when $minC$ increases above 1100. Similarly, for the CDC15 dataset, RSM is faster when $minC$ is less than 1100. This is due to the underlying working strategies of RSM and CubeMiner. As we know, the number of cutters in CubeMiner has an important effect on the tree's depth, and hence affects its performance. RSM mines on each representative slice, which has much fewer rows compared with the number of cutters in CubeMiner. That is, the datasets (representative slice) that RSM works on, is much smaller than the ones (whole dataset) that CubeMiner does. And the execution time of RSM is the sum of the execution time on each representative slice. This makes RSM efficient if the number of representative slices is not large. However, the number of representative slices increases very quickly with the increase of the dimension size to be enumerated, which limits the advantage of RSM to a great extent. That's why RSM runs faster when the enumerated dimension is very small but runs much slower as the smallest dimension grows. As we may see from RSM-H in Figure 3 (a), when the enumerated dimension has a size of 14, RSM-H performs worse than CubeMiner. And, as we shall see shortly, in the synthetic datasets where larger dimension size is used, this trend is more obvious.

Even when the enumerated dimension has a small size of 9 for RSM, with the increase in $minC$, CubeMiner catches up quickly. This is because CubeMiner directly works on the 3D dataset which prunes off the search space as soon as possible while RSM takes time in representative slice generation before performing space pruning.

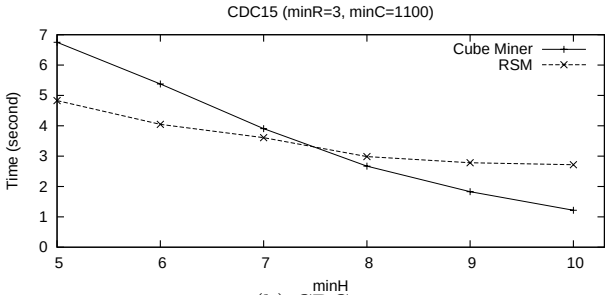
Next, we study the variation of $minH$, $minR$ on the two 3D datasets and set $minC = 1000$ for the Elutritration dataset and $minC = 1100$ for the CDC15 dataset. The $minC$ values are selected such that CubeMiner has a nearly similar but little longer processing time than RSM, to minimize the effects of $minC$ on the performance. Figure 4 and Figure 5 show the results respectively. The relative performance between RSM and CubeMiner remains largely the same for the same reasons given in the other experiments.

### 7.1.3 *Effect of Parallelism*

In this experiment, we study the effect of the number of processors on the processing time. The number of processors is varied from 1 to 32. We present the results on the CDC15 dataset. The results are shown in Figure 6. First, we observe that the parallel version of RSM-R outperforms the parallel version of CubeMiner. This is because, for this experiment, the experimental setup favors RSM-R, i.e., this is the setting where the uniprocessor RSM-R also outperforms CubeMiner (see Figure 3 where $minC = 1000$). Second, we note that as the degree of parallelism increases, the response time also decreases. Moreover, as in traditional parallel processing, there is a certain "optimal" number of processors beyond which additional parallelism leads to only marginal gain. In this experiment, for both schemes, the speedup is good for

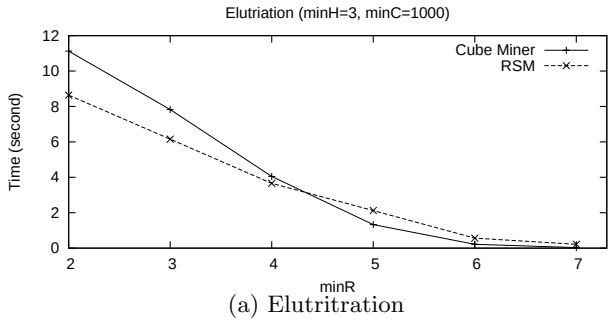


(a) Elutritration

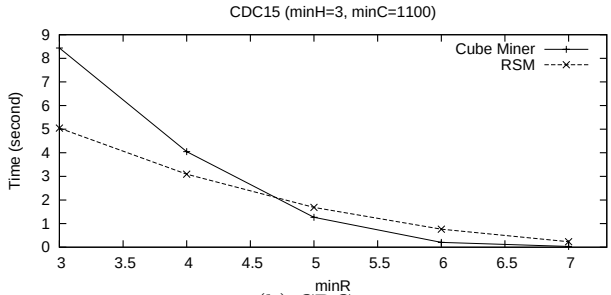


(b) CDC15

**Figure 4: Vary $minH$.**



(a) Elutritration



(b) CDC15

**Figure 5: Vary $minR$.**

upto 8 processors. Beyond 8 processors, the speedup starts to degrade.

## 7.2 Results on Synthetic Datasets

To study the scalability of our proposed schemes, we generate synthetic datasets using the IBM data generator. Since RSM's efficiency depends greatly on the size of the smallest dimension, in the first set of experiments, we study the effects of the size of smallest dimension on the execution time. We experiment on seven syntactic datasets with 30% density (percentage of cells with value one), 20 rows, 1000 columns, and the number of heights varied from 8 to 20. We set $minH = minR = 3$, and $minC = 30$ for all the experiments. Figure 7 shows the execution time in logarithm (second) scale. We see that the execution time of RSM and CubeMiner increase with increasing height number. We also observe that RSM's execution time increases much faster as the size of the heights increases. For larger datasets, CubeMiner is clearly much more efficient than RSM.

To study the scalability on large dataset, we generate syntactic datasets with 100 heights, 100 rows, 10000 columns, and 10% density. We study the execution time of RSM and CubeMiner with the variation of $minH$, $minR$, and $minC$. RSM failed to finish processing after long hours even on very high support constraints, which is incomparable to CubeMiner. Even its parallel version takes longer time than CubeMiner. This is because, with 100 heights, the number of slices to be enumerated is very large. Hence, we only report the execution time of CubeMiner and its parallel version P-CubeMiner with 8 processors (the "optimal" number) in Figure 8. From the results, we can confirm that (for the dataset used) 8 processors offer very good speedup. Moreover, we note that the parallel version of CubeMiner can reduce the computational cost significantly.

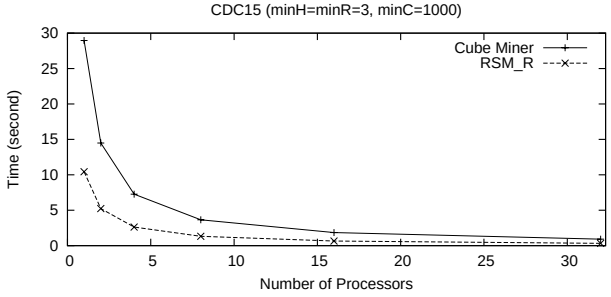



**Figure 6: Vary Number of Processors.**

From the experiments on synthetic datasets, we see that CubeMiner scales well on large datasets while RSM works efficiently only on datasets with a small size in one dimension.

# 8. CONCLUSION

In this paper, we have generalized 2D frequent closed pattern mining into 3D context. We defined the model of 3D frequent closed pattern – Frequent Closed Cube (FCC). We proposed two schemes to mine FCCs - while the Representative Slice Mining framework (RSM) enables us to reuse existing 2D frequent closed pattern mining algorithm, CubeMiner operates on the 3D space directly. We also presented parallel versions of the two schemes. We conducted extensive performance study on both real and synthetic datasets. Our results showed that both schemes can mine FCC efficiently, in particular, CubeMiner is superior for large datasets, while RSM performs best when one of the dimensions is small. Moreover, the parallel versions of both schemes can further reduce the computation time signifi-

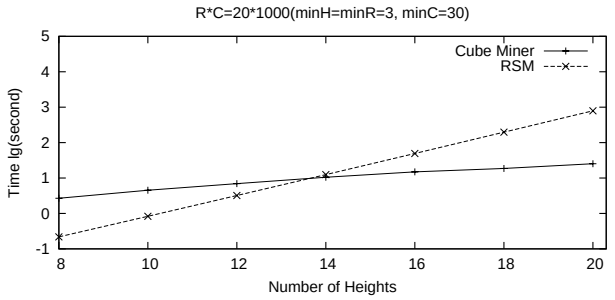
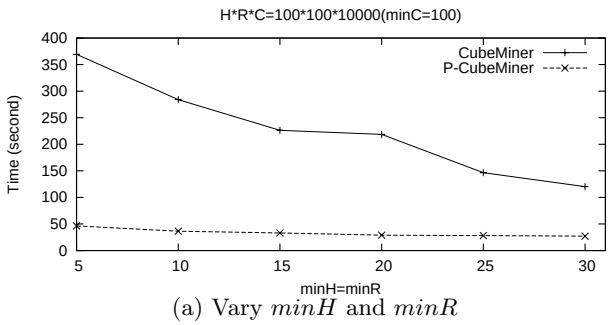
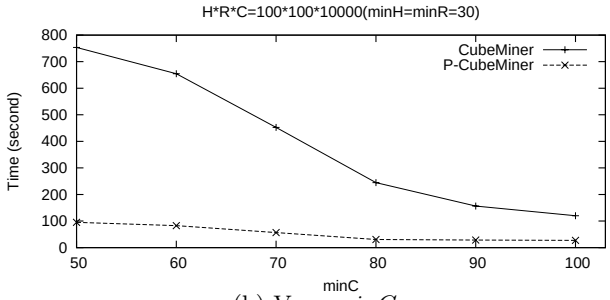



**Figure 7: Vary Size of Height Dimension.**




(a) Vary $minH$ and $minR$




(b) Vary $minC$

**Figure 8: Vary $minH$, $minR$ and $minC$.**

cantly. As future research, we plan to study 3D association rule analysis and classifier based on frequent closed cubes.

# 9. REFERENCES


[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases(VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE'95*, pages 3–14, Taibei, Taiwan, March 1995.

[3] J. Besson, C. Robardet, and J.-F. Boulicaut. Constraint-based mining of formal concepts in transactional data. In *PAKDD'04*, pages 615–624, Sydney, Australia, May 2004.

[4] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE'01*, pages 443–452, Heidelberg, Germany, April 2001.

[5] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE'99*, pages 106–115, Sydney, Australia, April 1999.

[6] D. Jiang, J. Pei, M. Ramanathany, C. Tang, and A. Zhang. Mining coherent gene clusters from gene-sample-time microarray data. In *SIGKDD'04*, pages 430–439, Seattle, Washington, USA, August 2004.

[7] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. In *Data Mining and Knowledge Discovery*, pages 259–289, 1997.

[8] F. Pan, G. Cong, and A. K. H. Tung. Carpenter: Finding closed patterns in long biological datasets. In *SIGKDD'03*, pages 637–642, Washington, DC, USA, August 2003.

[9] F. Pang, A. K. H. Tung, G. Cong, and X. Xu. Cobbler: Combining column and row enumeration for closed pattern discovery. In *SSDBM*, pages 21–30, Santorini Island, Greece, June 2004.

[10] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'99*, pages 398–416, Jerusalem, Israel, January 1999.

[11] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of 2000 ACMSIGMOD Int. Workshop Data Mining and Knowledge Discovery*, May 2000.

[12] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. In *CIKM'01*, pages 81–88, Atlanta, Georgia, November 2001.

[13] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *VLDB'98*, pages 368–379, New York City, USA, August 1998.

[14] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *SIGMOD'00*, pages 22–23, Dallas, TX, May 2000.

[15] P. T. Spellman, G. Sherlock, , and etc. Comprehensive identication of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. In *Molecular Biology of the Cell*, pages 3273–3297, December 1998.

[16] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *SIGKDD'03*, pages 236–245, Washington, D.C., August 2003.

[17] M. Zaki and C. Hsiao. CHARM: An efficient algorithm for closed association rule mining. In *SDM'02*, Arlington, VA, USA, April 2002.

[18] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD'97*, pages 283–286, Newport Beach, CA, August 1997.

[19] L. Zhao and M. J. Zaki. tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In *SIGMOD'05*, pages 694–705, Baltimore, Maryland, USA, June 2005.