Mining Bucket Order-Preserving SubMatrices in Gene Expression Data

Qiong Fang, Wilfred Ng, Jianlin Feng, Yuliang Li

Abstract—The Order-Preserving SubMatrices (OPSMs) are employed to discover significant biological associations between genes and experiment conditions. Herein, we propose a new relaxed OPSM model by considering the linearity relaxation, which is called the *Bucket OPSM (BOPSM)* model. An efficient method called APRIBOPSM is developed to exhaustively mine such BOPSM patterns. We further generalize the BOPSM model by incorporating the similarity relaxation strategy. We develop a generalized BOPSM model called *GeBOPSM* and adopt a pattern growing method called SEEDGROWTH to mine GeBOPSM patterns. Informally, the SEEDGROWTH algorithm adopts two different growing strategies on rows and columns in order to expand a seed BOPSM into a maximal GeBOPSM pattern. We conduct a series of experiments using both synthetic and biological datasets to study the effectiveness of our proposed relaxed models and the efficiency of the relevant mining methods. The BOPSM model is shown to be able to capture the characteristics of noisy OPSM patterns, and is superior to the strict counterparts. APRIBOPSM is also significantly more efficient than *OPC-Tree*, which is the state-of-the-art OPSM mining method. Compared to all the current relaxed OPSM models, the GeBOPSM model achieves the best performance in terms of the number of mined quality patterns.

Index Terms—order-preserving submatrix, biclustering, bucket order, linearity relaxation, similarity relaxation, OPSM

1 INTRODUCTION

In gene expression analysis, the *Order-Preserving SubMatrices* (OPSMs) are employed to discover significant biological associations between genes and experiment conditions. Mining OPSMs has been extensively studied as a biclustering problem in the area of gene expression analysis [2], [11], [20], [5], [4], [6]. The gene expression data are usually presented as a matrix in which the rows correspond to a set of genes, the columns correspond to a set of experiment conditions, and the entries represent the expression levels of the genes under the conditions. The OPSM model, first proposed by Ben-Dor et al. [2], aims to capture the fact that the expression levels of a set of genes follow the same trend under a set of conditions, but show no obvious correlation under other conditions. Specifically, in a gene expression matrix, an OPSM consists of a subset of genes and a subset of conditions such that the expression levels of every gene induce the same linear order of the conditions. This linear order of the conditions represents the *consensus trend* that all the genes in an OPSM follow.

Consider the matrix shown in Table 1 and the entries highlighted in bold font in the submatrix (P, Q) with $P = \{g_1, g_2, g_4\}$ and $Q = \{t_1, t_2, t_3, t_4\}$. For any gene g_i in P, we order its expression levels under conditions in Q in ascending order and then replace the values by their corresponding condition labels. Hence, all the genes in P induce the same linear order of Q, i.e., $[t_4 \succ t_2 \succ t_1 \succ t_3]$ and in this case, we say that the submatrix (P, Q) is an OPSM.

TABLE 1
A Gene Expression Matrix Example

Genes \ Conditions	t_1	t_2	t_3	t_4	t_5
g_1	**9**	**7**	**16**	**2**	20
g_2	**10**	**8**	**20**	**6**	1
g_3	5	16	13	4	10
g_4	**6**	**3**	**9**	**2**	5

The adoption of the OPSM model promotes the discovery of interesting biological associations between genes and experiment conditions in gene expression analysis. However, as already pointed out by Ben-Dor et al. [2], the OPSM model may be too restrictive in practice. For example, in cancer genome datasets, correlated genes may have very similar expression values under the same stage that consists of a subset of conditions (or a bucket of conditions), but exhibit some consensus trend from one stage to another stage during the progression of the disease. In this case, the strict OPSM model is not effective. In addition, other sources of noise such as sample contamination, experimental error, and the precision limits of instruments are not uncommon in bioinformatics data. Thus, more recent focus turns to relax the OPSM model in order to allow more general order and tolerate noise in data [5], [20].

- *Qiong Fang, Wilfred Ng and Yuliang Li are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. E-mail:fang@cse.ust.hk, wilfred@cse.ust.hk, cs_lyxab@stu.ust.hk*
- *Jianlin Feng is with the School of Software, Sun Yat-Sen University, Guangzhou, China. E-mail:fengjlin@mail.sysu.edu.cn*

Corresponding Author: Jianlin Feng. Phone and Fax: 86-20-39943152. Email: fengjlin@mail.sysu.edu.cn

There are two basic principles that can be adopted to relax the OPSM model.

1) The induced orders of all the genes in the relaxed OPSM patterns are allowed to be only similarly ordered instead of identically ordered. We call this relaxation strategy *similarity relaxation*.
2) The induced orders of the genes are allowed to be an identical bucket order, where each bucket contains a set of conditions. We call this relaxation strategy *linearity relaxation*.

In this paper, we first consider the linearity relaxation strategy, and propose a novel *Bucket OPSM* (BOPSM) model. The BOPSM model requires that all the genes in a BOPSM pattern support a consensus *bucket order* of a set of conditions, in the sense that the condition values of a gene in different buckets should maintain the ordering relationship between the buckets, and the condition values in the same bucket should be similar enough.

Then, we further improve the BOPSM model by considering the similarity relaxation strategy, and propose a relaxed BOPSM model, called *Generalized BOPSM* (GeBOPSM). The GeBOPSM model uses a backbone bucket order to capture the underlying consensus trend that the genes in a GeBOPSM follow. Each gene in the GeBOPSM should support a bucket order such that the similarity between the bucket order and the backbone bucket order be large enough.

While mining OPSM patterns is known to be an NP-complete problem [2], mining relaxed OPSM patterns is even more difficult. To tackle this problem, we propose an efficient APRIBOPSM method that mines BOPSM patterns. The APRIBOPSM method adopts the Apriori-based framework, and makes use of a new *BucketPrefixTree* structure to mine BOPSM patterns. Then, we propose a new GeBOPSM mining method called SEEDGROWTH that takes BOPSM patterns as seeds, and expands them into maximal GeBOPSMs.

In summary, the main contributions of this paper are twofold.

1) We propose a novel BOPSM model and develop an efficient breadth-first method called APRIBOPSM, which exhaustively mines BOPSM patterns. Empirical studies show that the adoption of the BOPSM model improves the quality of the mined patterns compared to the strict OPSM model. The APRIBOPSM method is also significantly more efficient than the state-of-the-art OPSM mining method *OPC-Tree*, when it is used to mine strict OPSM patterns. This shows the robustness of the APRIBOPSM algorithm.
2) We further generalize the BOPSM model and propose the GeBOPSM model. Experiments show that, compared to all current relaxed OPSM models, the GeBOPSM model better captures the characteristics of noise-contaminated OPSM patterns in real gene expression data. We develop an efficient mining method called SEEDGROWTH, which is able to mine sufficient GeBOPSM patterns by growing seed BOPSMs into maximal GeBOPSMs.

The organization of the rest of this paper is as follows. We introduce the related work in Section 2. Section 3 introduces the basic concepts and notations. In Section 4, a new BOPSM model together with its mining method are given. In Section 5, we introduce the GeBOPSM model and the method for mining such patterns. Experiments on both synthetic datasets and a real gene expression dataset are presented in Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK

The problem of mining order-preserving submatrices (OPSM), which can be viewed as a biclustering problem, has been extensively studied in the area of gene expression analysis [2], [11], [20], [5], [4], [6]. The concept of biclustering for gene expression analysis was firstly introduced by Chen and Church [3]. There follow different formulations of biclustering problems in the context in order to capture various biological associations among correlated genes and experiment conditions [12], [9], [19], [2], [13], [10], [7], [8].

The OPSM model was originally proposed by Ben-Dor et al. [2], which captures the fact that the expression levels of a set of genes exhibit similar trend under a set of experiment conditions. Experiments in [16] show that, compared to several other types of biclustering models such as CC [3] and Bimax [16], the OPSM model promotes the discovery of a larger fraction of biologically significant patterns based on a real biological dataset. However, it has also been recognized that the OPSM model may be too strict to be practical, since the real gene expression data are noisy [2], [20], [5]. Thus, different relaxation approaches on the model are studied.

The AOPC model [20] relaxes the condition that all the rows in an OPSM should induce the same linear order of columns, and it requires only that a pre-specified fraction of rows induce the same linear order, while the induced orders of other rows only need to be "similar enough". The ROPSM model proposed by Fang et al. [5] is a further relaxation of the AOPC model also based on "similarity". The ROPSM model uses the backbone order to capture the consensus trend of an ROPSM pattern, and only requires that the induced orders of all the rows in the pattern be similar enough to the backbone order. Basically, both the AOPC model and the ROPSM model only consider the similarity relaxation approach.

Mining OPSM patterns is shown to be an NP-complete problem in [2], and mining relaxed OPSMs accordingly becomes more difficult. Ben-Dor et al. [2] proposed a model-based method, which aims to

mine the best OPSM in terms of the statistical significance. Their method keeps a limited number of partial models which are smaller OPSMs, and then expands the partial models into larger OPSMs. Their method, however, is heuristic-based, and the significance of the mined OPSM is very sensitive to the selection of the partial models. Later, Liu et al. [11] proposed a tree-based OPSM mining method, called *OPC-Tree*, which can exhaustively mine all the OPSMs that satisfy some size thresholds. Their method enumerates all possible linear orders of columns in the depth-first manner with some pruning techniques being adopted. However, when the number of columns increases, the size of the tree grows extremely large, which greatly degrades the performance of pattern mining. Gao et al. proposed a KiWi framework in [6], which aims to mine "twig OPSMs", that are characterized by containing a large number of columns and very few rows. The framework expands a limited number of linear orders of columns in a breadth-first manner, and applies very strong conditions for pruning those linear orders that are less likely to grow into twig OPSMs. Their method is shown to be efficient but valid twig OPSMs may also get pruned.

The AOPC mining method proposed in [20] takes a set of OPSMs as input, and merges pairs of OPSMs into AOPCs in a greedy way until no more AOPCs can be generated. The ROPSM mining method proposed in [5] similarly takes a set of OPSMs as input. Instead of merging OPSMs, it expands those seed OPSMs by adopting different growing strategies until maximal ROPSMs are reached.

It is also worth noting that, if we transform the input matrix into a set of attribute (i.e., column label) sequences ordered by their values in a row, and view the collection of these sequences as a transaction database, the problem of mining OPSM patterns is actually convertible to the problem of mining frequent sequential patterns. Therefore, the frequent pattern mining methods can also be adopted to mine OPSMs. However, the OPSM mining problem has some unique properties which render the frequent pattern mining methods not efficient. First, each attribute item appears at most once in each transaction. Second, since the gene expression matrix is usually very dense, the transformed transactions are also dense, which may greatly degrade the performance of common frequent sequential pattern mining methods.

PrefixSpan [14], [15] is a well-known efficient frequent sequential mining method, and the *OPC-Tree* method improves the basic techniques of PrefixSpan and takes into account the characteristics of the OPSM mining problem (cf. [11] shows that the *OPC-Tree* outperforms PrefixSpan in mining OPSMs). Agrawal et al. proposed an Apriori-based sequential mining method in [1], [18], and the sequential patterns they mine can be regarded as "bucket orders". They adopted a preprocessing step which first mines all frequent itemsets (i.e., as buckets) and then transforms the frequent itemsets into new single items. Their Apriori-based mining method only mines sequential linear patterns from the transformed data. However, their method cannot be used to mine our BOPSM patterns. One reason is that a gene expression matrix is usually very dense, and thus the number of such frequent itemsets grows exponentially large. Another reason is that our BOPSM model adopts multiple thresholds to control the quality of BOPSMs, which cannot be incorporated into their method. These fundamental differences will be elaborated in Section 4.

3 Preliminaries

In this section, we introduce some concepts and notations that are used throughout the paper.

We use $M(G,T)$ to denote an input gene expression matrix, where G is the set of rows representing genes and T is the set of columns representing conditions or items. An entry of the matrix, denoted as $M(g,t)$, is the expression level of gene g under condition t.

Given a set of items $Q = \{q_1, \ldots, q_m\}$, a *linear order* of Q is represented as $[q_{i_1} \succ q_{i_2} \succ \cdots \succ q_{i_m}]$, where $\langle i_1, \ldots, i_m \rangle$ is a permutation of $\{1, \ldots m\}$, and the order relation " $\succ$ " satisfies the criteria of anti-symmetry, transitivity and linearity. A *bucket order* of Q is represented as $\tau_Q = [Q^{(1)} \succ Q^{(2)} \succ \cdots \succ Q^{(r)}]$, where $Q^{(i)} \subseteq Q$, $\cup_{i=1}^{r} Q^{(i)} = Q$, and $Q^{(i)} \cap Q^{(j)} = \phi$ for all i and j with $i \neq j$. When all buckets contain only one item, the bucket order is reduced to a linear order, and thus the linear order is a special case of the bucket order. We use Greek letters such as τ and π to represent orders. A gene g is said to *maintain* a bucket order τ_Q if the expression levels of g under all the conditions in $Q^{(i)}$ are less than the expression levels of g under all the conditions in $Q^{(i+1)}$.

Given two bucket orders $\tau_1 = [Q_1^{(1)} \succ \cdots \succ Q_1^{(r)}]$ and $\tau_2 = [Q_2^{(1)} \succ \cdots \succ Q_2^{(s)}]$, we say that τ_2 is the *sub-order* of τ_1 if there exists $1 \leq i_1 < i_2 < \cdots < i_s \leq r$ such that $Q_2^{(k)} \subseteq Q_1^{(i_k)}$ with $1 \leq k \leq s$.

4 The Bucket OPSM

In this section, we first define the Bucket OPSM (BOPSM) model, and then introduce an algorithm called AprioriBOPSM that mines BOPSM patterns.

4.1 The BOPSM Model

We now propose a new relaxed OPSM model called the *Bucket OPSM* model, or the BOPSM model for short, as follows.

Definition 4.1. ***(Bucket OPSM (Preliminary Version))*** *Given a submatrix (P,Q) of $M(G,T)$, (P,Q) is said to be a BOPSM if there exists a bucket order τ_Q such that all the genes in P maintain τ_Q.*

	a	b	c	d	e	f	g
g_1	2.4	2.7	3	10	10.6	10.2	15
g_2	5.8	5.6	6	8.9	8.7	9.0	12
g_3	0.4	0.7	1	4.3	4.6	4.6	8

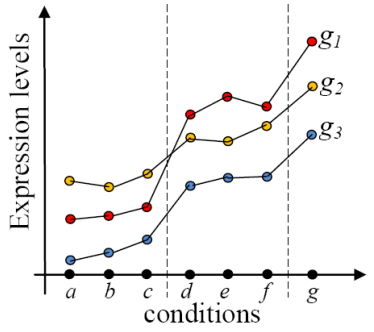


Fig. 1. A Running Example of BOPSM

Note that in Definition 4.1, within each bucket $Q^{(i)}$, no restrictions are placed on the ordering of the expression levels. Informally, the bucket order τ_Q represents the consensus trend that all the genes in a BOPSM should follow. We call τ_Q the *backbone (bucket) order* of the BOPSM (P, Q). Since a BOPSM (P, Q) is always associated with a backbone order τ_Q, we usually denote the BOPSM by $(P, Q : \tau_Q)$.

Figure 1 shows a BOPSM example, where the table is a submatrix (P, Q) with $P = \{g_1, g_2, g_3\}$ and $Q = \{a, b, c, d, e, f, g\}$, and the diagram is the graphical illustration of the table. We can check that all the three genes in P maintain a bucket order $\tau_Q = [\{a, b, c\} \succ \{d, e, f\} \succ \{g\}]$, and thus $(P, Q : \tau_Q)$ is a BOPSM.

Notably, a gene may maintain more than one possible bucket order for a given set of conditions. For example, all the three genes in Figure 1 also maintain another bucket order $\tau'_Q = [\{a, b\} \succ \{c, d, e, f\} \succ \{g\}]$. Clearly, the bucket order τ_Q better captures the progression from one stage, which consists of a set of conditions, to another stage. This is due to the fact that, for all the genes in P, the expression values under the conditions in the same bucket of τ_Q are more similar than those of τ'_Q while the difference between the expression values under the conditions in different buckets of τ_Q are larger than those of τ'_Q. This observation motivates us to improve Definition 4.1.

We need two concepts called *Intra-bucket Difference* vector and *Inter-bucket Gap* vector in order to formulate an improved version of Definition 4.1. We let $\tau = [Q^{(1)} \succ \cdots \succ Q^{(r)}]$ and assume a row g maintains τ.

- The *intra-bucket difference vector* of g (or simply *DVector* of g) with respect to τ, denoted as $\Delta(g, \tau)$, is an r-dimensional vector $\Delta(g, \tau) = \langle \delta_1(g, \tau), \ldots, \delta_r(g, \tau) \rangle$, where $\delta_i(g, \tau)$ is given by

$$\delta_i(g, \tau) = | \max_{q \in Q^{(i)}} M(g, q) - \min_{q \in Q^{(i)}} M(g, q) |.$$

- The *inter-bucket gap vector* of gene g (or simply *GVector* of g) with respect to τ, denoted as $\Gamma(g, \tau)$, is an $(r-1)$-dimensional vector $\Gamma(g, \tau) = \langle \gamma_1(g, \tau), \ldots, \gamma_{r-1}(g, \tau) \rangle$, where $\gamma_i(g, \tau)$ is given by

$$\gamma_i(g, \tau) = | \min_{q \in Q^{(i+1)}} M(g, q) - \max_{q \in Q^{(i)}} M(g, q) |.$$

Referring to the table in Figure 1, the *DVector* and *GVector* of g_1 with respect to τ_Q and τ'_Q are respectively $\Delta(g_1, \tau_Q) = \langle 0.6, 0.6, 0 \rangle$, $\Gamma(g_1, \tau_Q) = \langle 7.0, 4.4 \rangle$, $\Delta(g_1, \tau'_Q) = \langle 0.3, 7.6, 0 \rangle$ and $\Gamma(g_1, \tau'_Q) = \langle 0.3, 4.4 \rangle$. We can see that all the elements in $\Delta(g_1, \tau_Q)$ are small while the elements of $\Gamma(g_1, \tau_Q)$ are comparatively large. However, we find a relatively large element in $\Delta(g_1, \tau'_Q)$ (i.e. 7.6) or a relatively small element in $\Gamma(g_1, \tau'_Q)$ (i.e. 0.3). Similar situations can be found in the *DVectors* and *GVectors* of g_2 and g_3.

We now revise Definition 4.1 in order to mine patterns where all the associated genes have sufficiently small intra-bucket difference and sufficiently large inter-bucket gap with respect to the backbone order.

Definition 4.2. ***(Bucket OPSM (Improved Version))*** *Given a submatrix (P, Q), a difference threshold d_{max} and a gap threshold g_{min}, (P, Q) is said to be a BOPSM if there exists a bucket order τ_Q such that, for all $g_i \in P$,*

1) *g_i maintains the bucket order τ_Q, and*
2) *$\delta_j(g_i, \tau_Q) \leq d_{max}$ and $\gamma_k(g_i, \tau_Q) \geq g_{min}$ for all $\delta_j(g_i, \tau_Q) \in \Delta(g_i, \tau_Q)$ and $\gamma_k(g_i, \tau_Q) \in \Gamma(g_i, \tau_Q)$.*

Given a bucket order τ_Q, if a row g maintains τ_Q and its *DVector* and *GVector* both satisfy the conditions, we say row g *supports* the bucket order τ_Q, or g is a *supporting row* of τ_Q. Continuing the example in Figure 1, if we set the thresholds d_{max} and g_{min} both to be 1.0, $(P, Q : \tau_Q)$ is still a valid BOPSM while $(P, Q : \tau'_Q)$ is not any more.

From now on, we refer to Definition 4.2 the formal meaning of the BOPSM model in our subsequent discussion.

In the BOPSM model, two scalar thresholds d_{max} and g_{min} are used to control the quality of BOPSM patterns. One simple choice for them is the average gap. That is, for every gene, we get the induced linear order of conditions, and compute the average of the gaps between the expression values under every pair of adjacent conditions in the order. We can set the thresholds to be the global average gap of all the genes. Another more sophisticated choice is, considering the scaling of the expression levels of different genes, we can set a specific threshold for a particular gene by using its own average gap. Our proposed techniques can be straightforwardly extended to this general settings by replacing the scalar versions of d_{max} and g_{min} with their respective vector versions.

The BOPSM model holds a useful anti-monotonic property, which will be used to develop an efficient algorithm for mining BOPSM patterns.

Theorem 4.3. ***(Anti-Monotonicity)*** *If a row g supports a bucket order τ_Q, g also supports all the sub-orders of τ_Q.*

Proof: Suppose there are two bucket orders $\tau_1 = [Q_1^{(1)} \succ \cdots \succ Q_1^{(r)}]$ and $\tau_2 = [Q_2^{(1)} \succ \cdots \succ Q_2^{(s)}]$, and τ_2 is the sub-order of τ_1. Thus, we have $s \leq r$ and there exist $1 \leq i_1 < i_2 < \cdots < i_s \leq r$ such that $Q_2^{(k)} \subseteq Q_1^{(i_k)}$ for $1 \leq k \leq s$.

Assume that g maintains τ_1 and the vectors $\Delta(g, \tau_1)$ and $\Gamma(g, \tau_1)$ respectively satisfy the following threshold conditions: $\delta_j(g, \tau_1) \leq d_{max}$ for $1 \leq j \leq r$, and $\gamma_k(g, \tau_1) \geq g_{min}$ for $1 \leq k \leq (r-1)$.

Next, we show that g also maintains τ_2, and $\Delta(g, \tau_2)$ and $\Gamma(g, \tau_2)$ also satisfy the threshold conditions.

1) Since g maintains τ_1, it follows that

$$\min_{q \in Q_1^{(k+1)}} M(g,q) \geq \max_{q \in Q_1^{(k)}} M(g,q) \text{ for } 1 \leq k < r.$$

Then, because $Q_2^{(k)} \subseteq Q_1^{(i_k)}$ holds for all k, we have

$$\begin{aligned} \min_{q \in Q_2^{(k+1)}} M(g,q) &\geq \min_{q \in Q_1^{(i_{k+1})}} M(g,q) \\ &\geq \max_{q \in Q_1^{(i_k)}} M(g,q) \geq \max_{q \in Q_2^{(k)}} M(g,q) \end{aligned}$$

Thus, g also maintains τ_2.

2) For the *DVector* $\Delta(g, \tau_2)$, its k-th element is

$$\begin{aligned} \delta_k(g,\tau_2) &= |\max_{q \in Q_2^{(k)}} M(g,q) - \min_{q \in Q_2^{(k)}} M(g,q)| \\ &\leq |\max_{q \in Q_1^{(i_k)}} M(g,q) - \min_{q \in Q_1^{(i_k)}} M(g,q)| \leq d_{max} \end{aligned}$$

For the *GVector* $\Gamma(g, \tau_2)$, its k-th element is

$$\begin{aligned} \gamma_k(g,\tau_2) &= |\min_{q \in Q_2^{(k+1)}} M(g,q) - \max_{q \in Q_2^{(k)}} M(g,q)| \\ &\geq |\min_{q \in Q_1^{(i_{k+1})}} M(g,q) - \max_{q \in Q_1^{(i_k)}} M(g,q)| \\ &\geq |\min_{q \in Q_1^{(i_{k+1})}} M(g,q) - \max_{q \in Q_1^{(i_{k+1}-1)}} M(g,q)| \geq g_{min} \end{aligned}$$

Thus, it follows that $\Delta(g, \tau_2)$ and $\Gamma(g, \tau_2)$ satisfy the threshold conditions.

□

It is worth noting that, if we set the threshold d_{max} to be 0, and g_{min} to be some positive value smaller than all the gaps in all the induced orders of rows, the mined BOPSM patterns are actually OPSM patterns. We also need to include two size thresholds to guarantee that sufficiently large and thus significant patterns can be obtained, since a consensus is that, given a pattern with a fixed number of columns (or rows), a larger number of rows (or columns) usually leads to more significance [6].

We now formalize the BOPSM mining problem as follows.

Definition 4.4. ***(The BOPSM Mining Problem)*** *Given a matrix $M(G,T)$, the difference threshold d_{max}, the gap threshold g_{min}, the column threshold c_{min} and the row threshold r_{min}, we aim to mine from M all the valid BOPSMs that contain at least c_{min} columns and r_{min} rows.*

Algorithm 1: APRIBOPSM

Input: Matrix $M(G,T)$, d_{max}, g_{min}, c_{min}, r_{min}

1. $\mathcal{F}_1$ ={size-1 frequent bucket orders};
2. **for** $(k = 2; \mathcal{F}_{k-1} \neq \phi; k++)$ **do**
3. $\quad \mathcal{C}_k$= GENCAND($\mathcal{F}_{k-1}$) ;
4. $\quad$ COUNTSUP($M, \mathcal{C}_k$);
5. $\quad \mathcal{F}_k = \{\tau \mid \tau \in \mathcal{C}_k, supp(\tau) \geq r_{min}\}$;
6. $\quad$ **if** $k \geq c_{min}$ && $\mathcal{F}_k \neq \phi$ **then**
7. $\quad\quad$ Output BOPSMs ;
8. **end**

4.2 Mining BOPSM Patterns

Given a matrix $M(G,T)$, a naïve way to mine BOPSM patterns can be carried out as follows: for every set of columns Q with $Q \subseteq T$ and $|Q| \geq c_{min}$, and for every possible bucket order τ_Q, we simply check all the rows in G to see if they support τ_Q or not. Let P be the set of supporting rows of τ_Q. If the size of P is no less than r_{min}, (P, Q) is a valid BOPSM pattern with τ_Q as the backbone order. However, such an exhaustive checking is apparently infeasible, since the number of such bucket orders is prohibitively large, especially when the number of columns in T is large.

The anti-monotonic property of the BOPSM established in Theorem 4.3 is useful in developing a more efficient approach to mining BOPSM patterns. Before searching the supporting rows of a bucket order, we can first check that whether all its sub-orders are supported by at least r_{min} rows. If not, it is already confirmed that this bucket order does not lead to a valid BOPSM pattern. This idea motivates us to develop an Apriori-based framework to mine BOPSM patterns, as detailed in Algorithm 1.

In Algorithm 1, we define a bucket order τ containing k columns as a *size-k bucket order*. In addition, if τ is supported by at least r_{min} rows, we say that τ is a *frequent bucket order*. We now use $\mathcal{C}_k$ and $\mathcal{F}_k$ to denote the size-k bucket order set and the size-k frequent bucket order set respectively. First, we find all size-1 frequent bucket orders which are simply all the single columns (Line 1). Then, the GENCAND procedure (Algorithm 2) is invoked to generate size-k candidate bucket orders from the set of size-$(k-1)$ frequent bucket orders (Line 3). The number of supporting rows for each candidate bucket order is then counted by the COUNTSUP procedure (Line 4). The size-k candidate bucket orders that are supported by at least r_{min} rows form the size-k frequent bucket order set (Line 5). Finally, we output all the BOPSM patterns $(P, Q : \tau_Q)$, where τ_Q is a frequent bucket order with size at least c_{min}, Q is the set of columns involved in τ_Q, and P is the set of rows supporting τ_Q.

4.2.1 BucketPrefixTree

We develop a *BucketPrefixTree* (*BPTree* for short) structure to organize the candidate bucket orders and to help count their supporting rows. Given a set of

	Candidate bucket orders
τ_1	$[\{t_1, t_2, t_3\} \succ \{t_4, t_5\}]$
τ_2	$[\{t_1, t_2\} \succ \{t_3\} \succ \{t_4, t_5\}]$
τ_3	$[\{t_2, t_3\} \succ \{t_1, t_4, t_5\}]$

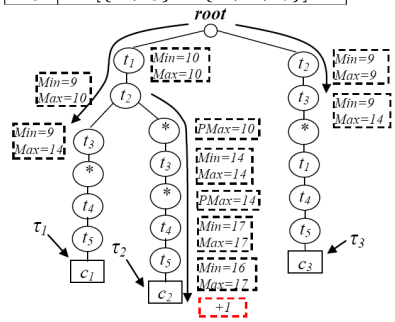


Fig. 2. A *BucketPrefixTree* Example

candidate bucket orders, we first linearize all the orders by ordering items within each bucket in some pre-specified manner (alphabetical or numerical). A special symbol such as " $*$ " is used to indicate the boundary between adjacent buckets. For example, the linearization of the bucket order $[\{t_2, t_1, t_3\} \succ \{t_4, t_5\}]$ is $[t_1 \succ t_2 \succ t_3 \succ * \succ t_4 \succ t_5]$. Then, we organize the linearizations of the bucket orders as a prefix-sharing tree, which is the *BPTree*.

Figure 2 shows a *BPTree* example, where three bucket orders τ_1, τ_2 and τ_3 are listed in the table, and the figure shows the corresponding *BPTree*. Since a path running from the root to a leaf node in the *BPTree* corresponds to one candidate bucket order (or its linearization), we associate with each leaf node a counting variable c_i that stores the number of supporting rows of the corresponding bucket order τ_i.

Our approach is Apriori-based, which shares the similar principle with the sequential pattern mining method proposed in [1]. However, we do not construct a new hash tree at each round for counting support values to save time. Instead, we only construct an initial *BPTree* at the first round based on the set of size-1 frequent bucket orders $\mathcal{F}_1$. Then in the subsequent rounds, the *BPTree* can be quickly updated by inserting new candidate bucket orders or deleting old ones. We denote the *BPTree* after the k-th round of candidate generation as $\mathcal{BPT}(\mathcal{C}_k)$, and denote the *BPTree* after the k-th round of supporting counting and infrequent candidate filtering as $\mathcal{BPT}(\mathcal{F}_k)$. We will show how *BPTree* helps generate the candidate bucket orders and count the supports in later sections.

4.2.2 Candidate Generation (GENCAND)

Algorithm 2 aims to generate size-k candidate bucket orders, i.e., $\mathcal{C}_k$, from $\mathcal{F}_{k-1}$ by making use of the *BPTree*. Depth-first traversal is conducted on $\mathcal{BPT}(\mathcal{F}_{k-1})$.

Given two size-$(k-1)$ bucket orders in $\mathcal{F}_{k-1}$, i.e., $\tau_1 = [Q_1^{(1)} \succ \cdots \succ Q_1^{(r)}]$ and $\tau_2 = [Q_2^{(1)} \succ \cdots \succ Q_2^{(s)}]$, they can be merged into a size-k bucket order only when the linearizations of τ_1 and τ_2 share a prefix that contains $(k-2)$ items. Therefore, we first traverse the tree to an item node p such that the bucket order

Algorithm 2: GENCAND

```
1.  while there are untraversed nodes in BPT(F_{k-1}) do
2.      Traverse to item node p s.t. |τ[p]| = k − 2;
3.      for item node t1, t2 of p's children do
4.          insert t2 as a child of t1;
5.      end
6.      if p has a boundary node q as child then
7.          for p's child item node t1 and q's child item
            node t2 do
8.              insert branch q → t2 under t1 if t2 ≠ t1;
9.          end
10.         for item nodes t1, t2 of q's children do
11.             insert branch q → t2 under t1;
12.             insert branch q → t1 under t2;
13.         end
14.     end
15. for all newly inserted leaf nodes t do
16.     if any size-(k − 1) sub-order of τ[t] ∉ F_{k-1} then
17.         undo insertion of node t;
18. end
```

$\tau[p]$, which corresponds to the path from $root$ to p, contains $(k-2)$ item nodes (Line 2). Then, there are three different cases for merging τ_1 and τ_2.

1) Case one: the last buckets of τ_1 and τ_2 both contain more than one item, and only the last items of these buckets are different in their linearizations. We then merge the last buckets of τ_1 and τ_2, and add a bucket order $[Q_1^{(1)} \succ \cdots \succ Q_1^{(r-1)} \succ Q_1^{(r)} \cup Q_2^{(s)}]$ to $\mathcal{C}_k$ (Lines 3 - 5).
2) Case two: p is the last but one item in the last bucket of τ_1, while it is the last item in the last but one bucket of τ_2. If the last item in the last bucket of τ_1 is different from the only item in the last bucket of τ_2, we add a bucket order $[Q_1^{(1)} \succ \cdots \succ Q_1^{(r)} \succ Q_2^{(s)}]$ into $\mathcal{C}_k$ (Lines 6 - 9).
3) Case three: p is the last item in the last but one bucket in both τ_1 and τ_2. The last buckets of both τ_1 and τ_2 then contain only one item. We add two size-k bucket orders, $[Q_1^{(1)} \succ \cdots \succ Q_1^{(r-1)} \succ Q_1^{(r)} \succ Q_2^{(s)}]$ and $[Q_1^{(1)} \succ \cdots \succ Q_1^{(r-1)} \succ Q_2^{(s)} \succ Q_1^{(r)}]$ into $\mathcal{C}_k$ (Lines 10 - 13).

Finally, for every newly generated size-k bucket order, we further check if all of its size-$(k-1)$ sub-orders are in $\mathcal{F}_{k-1}$. If not, this bucket order is surely not a frequent bucket order and thus can be discarded (Lines 15 - 18). The updated *BPTree* is $\mathcal{BPT}(\mathcal{C}_k)$.

Next, we prove the completeness of the GENCAND algorithm in order to confirm that all the size-k frequent bucket orders should be added into $\mathcal{C}_k$.

Theorem 4.5. *The* GENCAND *algorithm is complete.*

Proof: Let $\tau = [Q^{(1)} \succ \cdots \succ Q^{(r)}]$ be a size-k bucket order. It suffices to analyse the three cases concerning the size of the last two buckets $Q^{(r-1)}$ and $Q^{(r)}$ as follows. They cover all possible situations where a size-k frequent bucket order can be generated.

1) Suppose $|Q^{(r)}| > 1$. We denote the linearization of $Q^{(r)}$ as $\pi[Q^{(r)}] = [x_1 \succ \cdots \succ x_l]$ with $l > 1$. We can get two size-$(l-1)$ buckets, i.e., $Q_1^{(r)} = \{x_1, \ldots, x_{l-1}\}$ and $Q_2^{(r)} = \{x_1, \ldots, x_{l-2}, x_l\}$, and generate two size-$(k-1)$ bucket orders, $\tau_1 = [Q^{(1)} \succ \cdots \succ Q_1^{(r)}]$ and $\tau_2 = [Q^{(1)} \succ \cdots \succ Q_2^{(r)}]$. According to the anti-monotonic property, all the rows that support τ should definitely also support τ_1 and τ_2. In other words, if τ is a frequent bucket order, τ_1 and τ_2 must be frequent and in $\mathcal{F}_{k-1}$. Moreover, τ_1 and τ_2 satisfy the first merging case of the GENCAND algorithm, and thus they will be merged into τ, and τ is in $\mathcal{C}_k$.
2) Suppose $|Q^{(r)}| = 1$ and $|Q^{(r-1)}| > 1$. The linearization of $Q^{(r-1)}$ is denoted as $\pi[Q^{(r-1)}] = [x_1 \succ \ldots \succ x_l]$ with $l > 1$. By deleting the last item from $Q^{(r-1)}$, we get a new bucket $Q_1^{(r-1)} = \{x_1, \cdots, x_{l-1}\}$, and thus a new size-$(k-1)$ bucket order $\tau_1 = [Q^{(1)} \succ \cdots \succ Q_1^{(r-1)} \succ Q^{(r)}]$. Or, we delete the last bucket $Q^{(r)}$ and get another size-$(k-1)$ bucket order $\tau_1 = [Q^{(1)} \succ \cdots \succ Q^{(r-1)}]$. Similarly, if τ is frequent, τ_1 and τ_2 must also be frequent and exist in $\mathcal{F}_{k-1}$. According to the second merging case of the algorithm, τ will be generated by merging τ_1 and τ_2.
3) Suppose $|Q^{(r)}| = |Q^{(r-1)}| = 1$. We get two bucket orders, $\tau_1 = [Q^{(1)} \succ \cdots \succ Q^{(r-1)}]$ and $\tau_2 = [Q^{(1)} \succ \cdots \succ Q^{(r-2)} \succ Q^{(r)}]$. Similarly, if τ is a frequent bucket order, τ_1 and τ_2 must also be frequent and exist in $\mathcal{F}_{k-1}$. According to the third merging case of the algorithm, τ will be generated by merging τ_1 and τ_2. □

4.2.3 Support Counting (COUNTSUP)

The procedure COUNTSUP is used to count the supporting rows for each candidate in $\mathcal{C}_k$. Due to the space limit, we do not show the details of the algorithm but explain the underlying ideas below.

For a row g, we traverse the *BPTree* in the depth-first manner to find all the candidate bucket orders it supports. There are different operations to handle different types of nodes as follows.

1) **When an item node t is traversed**, we set its associated Min and Max variables respectively with the minimum and maximum values of the items in the current bucket encountered so far. Then, we respectively check whether the conditions $|t.Max - t.Min| \leq d_{max}$ and $|t.Min - p.PMax| \geq g_{min}$ are satisfied, where p is the boundary node of the current bucket and is just traversed. If both conditions are satisfied and node t is a leaf node, we increase the associated counting variable by 1, and traverse to the next unvisited node. If both conditions are satisfied but t is not a leaf node, we continue to traverse the children of t. Otherwise, we stop traversing along the current path, backtrack and traverse to next unvisited node.
2) **When a boundary node is traversed**, we set its associated $PMax$ variable as the maximum value of the items in the bucket just traversed.

Consider the *BPTree* in Figure 2 again. Suppose $g = \{t_1 : 10, t_2 : 9, t_3 : 14, t_4 : 17, t_5 : 16\}$, where the numbers after the colon are the condition values of g. The thresholds g_{min} and d_{max} are both set to be 2. We start from *root* and traverse along the leftmost path. When the node t_3 is met, the Min and Max values are 9 and 14. Since the condition that $|t_3.Max - t_3.Min| \leq d_{max}$ is not satisfied, we return to the parent node t_2 and visit its next unvisited child, which is a boundary node. The $PMax$ value of the boundary node is set to be the Max value of its parent, which is 10. Then, we continue to traverse the child of the boundary node all the way till t_5. We find that the two conditions are satisfied. Since t_5 is the leaf node, we increase its associated counting variable c_2 by 1, which means that row g supports the candidate bucket order τ_2. We keep traversing the *BPTree* until no more nodes need to be visited.

After traversing the *BPTree* for all the rows, we prune the tree by deleting the branches whose associated counting variables are smaller than r_{min}. These branches correspond to the candidate bucket orders that are supported by less than r_{min} rows, and the updated *BPTree* is $\mathcal{BPT}(\mathcal{F}_k)$.

5 THE GENERALIZED BOPSM

In this section, we relax the BOPSM model by adopting the similarity relaxation strategy. We propose the generalized BOPSM model (GeBOPSM) and present an efficient algorithm that mines GeBOPSM patterns.

5.1 The Generalized BOPSM Model

We first define an *LCS similarity* function to measure the similarity between two bucket orders.

Definition 5.1. ***(The LCS Similarity)*** *Given two bucket orders, $\tau_1 = [Q_1^{(1)} \succ \cdots \succ Q_1^{(r)}]$ and $\tau_2 = [Q_2^{(1)} \succ \cdots \succ Q_2^{(s)}]$, the LCS similarity between τ_1 and τ_2, denoted as $d_{LCS}(\tau_1, \tau_2)$, is given by*

$$d_{LCS}(\tau_1, \tau_2) = \frac{|LCS(\tau_1, \tau_2)|}{|T(\tau_1) \cup T(\tau_2)|},$$

where $|LCS(\tau_1, \tau_2)|$ is the number of items in the longest common sub-order *between τ_1 and τ_2, and $|T(\tau_1) \cup T(\tau_2)|$ is the number of items involved in τ_1 and τ_2.*

Using similar analysis in [5], the LCS similarity between two size-k bucket orders can also be computed in $O(k^2)$ time by dynamic programming.

Now, we formally define the generalized BOPSM model as follows.

Definition 5.2. ***(Generalized BOPSM (GeBOPSM))*** *Given a submatrix (P, Q), a similarity threshold α, a*

TABLE 2
A GeBOPSM Example

	a	b	c	d	e	f	g	h
g_1	2.5	2.7	5.0	10.1	10.6	10.2	15.6	15.0
g_2	6.3	6.2	6.4	8.9	6.7	9.0	11.3	10.7
g_3	0.3	0.7	1.1	4.3	4.6	4.6	8.7	9.1

(a) A submatrix (P, Q) with $P = \{g_1, g_2, g_3\}$ and $Q = \{a, \ldots, h\}$

$O(P,Q)$		d_{LCS}
o_1^Q	$[\{a,b\} \succ \{c\} \succ \{d,e,f\} \succ \{g,h\}]$	0.875
o_2^Q	$[\{a,b,c,e\} \succ \{d,f\} \succ \{g,h\}]$	0.875
o_3^Q	$[\{a,b,c\} \succ \{d,e,f\} \succ \{g,h\}]$	1.0
$\tau_Q = [\{a,b,c\} \succ \{d,e,f\} \succ \{g,h\}]$		

(b) The supported set $O(P, Q)$ and τ_Q

difference threshold d_{max}, and a gap threshold g_{min}, (P, Q) is said to be a GeBOPSM if there exists τ_Q such that for all $g_i \in P$,

1) *g_i supports a bucket order o_i^Q, and*
2) *the LCS similarity between o_i^Q and τ_Q is at least α.*

The bucket order τ_Q in Definition 5.2 captures the consensus trend that all the rows in a GeBOPSM pattern follow. We call τ_Q the *backbone (bucket) order* of the GeBOPSM and denote the GeBOPSM as $(P, Q : \tau_Q)$. A GeBOPSM pattern $(P, Q : \tau_Q)$ is said to be *maximal* if there does not exist any other GeBOPSM $(P', Q' : \tau_{Q'})$ such that $P \subseteq P'$, $Q \subseteq Q'$, o_i^Q is the sub-order of $o_i^{Q'}$ for all $g_i \in P$, and τ_Q is the sub-order of $\tau_{Q'}$.

We now introduce some terminologies for ease of discussion. If a row g_i supports a bucket order o_i^Q, and the LCS similarity between o_i^Q and τ_Q is no smaller than α, we say that g_i *α-supports* τ_Q. We call the set of bucket orders o_i^Q for all the rows in P the *supported set* of (P, Q), and denote this set by $O(P, Q)$.

Let us illustrate the GeBOPSM model by the submatrix (P, Q) shown in Table 2(a), where α is 0.8, and d_{max} and g_{min} are both 1.0. We can find $O(P, Q)$ as shown in Table 2(b) and $\tau_Q = [\{a, b, c\} \succ \{d, e, f\} \succ \{g, h\}]$ such that every row in P α-supports τ_Q. For example, g_1 supports o_1^Q with respect to g_{min} and d_{max}. Besides, one longest common sub-order between τ_Q and o_1^Q is $[\{a, b\} \succ \{d, e, f\} \succ \{g, h\}]$, and thus the LCS similarity between them is $\frac{7}{8} = 0.875$, which is larger than α. Therefore, $(P, Q : \tau_Q)$ is a GeBOPSM.

When setting α to be 1.0, the GeBOPSM model is reduced to the BOPSM model. On the other hand, when d_{max} is set to be 0, and g_{min} to be a very small positive value, the order o_i^Q is actually the induced linear order of row g_i on Q. If we also require τ_Q to be a linear order, the GeBOPSM model is reduced to the ROPSM model in [5]. As the ROPSM model is shown to be a generalization of the AOPC model, we thus claim that all the known relaxed OPSM models are actually the specializations of our GeBOPSM model.

Having defined the GeBOPSM model, we now formulate the GeBOPSM mining problem as follows:

Definition 5.3. ***(The GeBOPSM Mining Problem).*** *Given a matrix $M(G, T)$, and the similarity threshold α, the difference threshold d_{max}, the gap threshold g_{min}, the size thresholds c_{min} and r_{min}, we aim to mine from M the valid maximal GeBOPSM patterns.*

Notably, we do not aim to exhaustively mine all valid maximal GeBOPSM patterns, which is infeasible. However, we will demonstrate in next section that our proposed algorithm is capable of generating a substantial number of significant GeBOPSM patterns.

5.2 Mining GeBOPSM Patterns

One challenge of mining GeBOPSM patterns is that the GeBOPSM model does not enjoy the anti-monotonic property and thus it is difficult to develop an Apriori-based framework to mine GeBOPSM patterns. We now propose a two-phase approach to solve the mining problem, which has been successfully used for mining ROPSM patterns in our earlier work [5].

- First phase: we mine a set of seed BOPSMs by adopting the APRIBOPSM method as already discussed.
- Second phase: we adopt a pattern growing method called SEEDGROWTH to expand the seed BOPSMs into maximal GeBOPSM patterns.

We now detail the SEEDGROWTH algorithm.

5.3 SEEDGROWTH

The algorithm takes BOPSMs as seed patterns and expands them into maximal GeBOPSM patterns. The difficulty is that, within the GeBOPSM model, a row may support more than one bucket order. For example, if d_{max} is 3.0 and g_{min} is 1.0, g_1 in Table 2(a) supports both $[\{a, b\} \succ \{c\} \succ \{d, e, f\} \succ \{g, h\}]$ and $[\{a, b, c\} \succ \{d, e, f\} \succ \{g, h\}]$. Thus, the SEEDGROWTH algorithm not only needs to identify the backbone order but also needs to find the appropriate bucket orders supported by the rows (i.e., the supported set).

5.3.1 Column Expansion and Row Expansion

The SEEDGROWTH algorithm expands a seed BOPSM by rows and columns. We implement a greedy procedure called GRDYCOLEXP, which always takes the "best" possible column (in terms of some criterion) to expand the current pattern. The row expansion procedure is called EXHROWEXP, which scans the remaining rows that are not included in the current pattern and expands the pattern by those rows that β-support the backbone order having $\beta \geq \alpha$. The details of the GRDYCOLEXP and EXHROWEXP procedures are shown in Algorithms 3 and 4.

The GRDYCOLEXP algorithm takes a GeBOPSM pattern (initially a seed BOPSM) $(P, Q : \tau_Q)$ as input, together with the thresholds α, d_{max}, and g_{min}. For every remaining column t_j and every bucket order o_i in $O(P, Q)$, we first extend o_i by t_j and find the extended bucket orders from o_i. The set of extended

Algorithm 3: GRDYCOLEXP

Input: GeBOPSM $(P, Q : \tau_Q)$; α, d_{max} and g_{min}

1. **for** $t_j \in T - Q$ **do**
2. **for** o_i *in* $O(P, Q)$ **do**
3. $O_i = \text{EXTORDER}(o_i, t_j, d_{max}, g_{min})$;
4. **end**
5. $\tau_{Q\cup\{t_j\}} = \text{MEDBACKBONE}(\cup_i Q_i)$;
6. **for** $g_i \in P$ **do**
7. $\mu(g_i, t_j) = \max\{d_{LCS}(\tau_{Q\cup\{t_j\}}, o_k), o_k \in O_i\}$;
8. **end**
9. $\mu(t_j) = \min\{\mu(g_i, t_j), g_i \in P\}$;
10. **end**
11. Pick t_k s.t. $\mu(t_k) = \max\{\mu(t_j), \forall t_j \in T - Q\}$;
12. **if** $\mu^* = \mu(t_k) \geq \alpha$ **then**
13. Update τ_Q to $\tau_{Q\cup\{t_k\}}$; $Q = Q \cup \{t_k\}$;
14. Add o_k to $O(P, Q)$ such that $o_k = arg\max_{o_j}\{d_{LCS}(\tau_{Q\cup\{t_k\}}, o_j), \forall o_j \in O_i\}$;
15. Return *succeed*;
16. Return *fail*;

bucket orders, denoted as O_i, should still be supported by row g_i (Lines 2-4). Using the extended bucket orders of all the rows in P, we adopt the MEDBACKBONE method (discussed later) to compute the updated backbone order $\tau_{Q\cup\{t_j\}}$ (Line 5). Then, For each row g_i, we pick the extended bucket order among O_i, which maximizes the LCS similarity with the updated backbone order $\tau_{Q\cup\{t_j\}}$ (Lines 6-8). We use $\mu(t_j)$ to denote the smallest maximum LCS similarity among all the rows in P (Line 9). The column t_k that maximizes $\mu(t_k)$ is picked (Line 11). If $\mu(t_k)$ is no smaller than the similarity threshold α, we expand the current GeBOPSM by column t_k, and update the backbone order as well as the supported set $O(P, Q)$ of the updated GeBOPSM (Lines 12-15).

The EXTORDER procedure invoked at Line 3 extends the bucket order o_i supported by row g_i with a new column t_j. The requirement is that the extended bucket orders should still be supported by g_i and have the original o_i as a sub-order. Let us use the row g_1 in Table 2(a) to illustrate this point. Suppose g_1 supports a bucket order $o_1 = [\{a, b\} \succ \{d, f\} \succ \{g, h\}]$, and the thresholds d_{max} and g_{min} are respectively 3.0 and 2.0. When we extend o_1 with column c, c can either be inserted into the first bucket of o_1, or form a new bucket between the first and the second buckets of o_1. Thus, we get two extended bucket orders $[\{a, b, c\} \succ \{d, f\} \succ \{g, h\}]$ or $[\{a, b\} \succ \{c\} \succ \{d, f\} \succ \{g, h\}]$.

Actually, in general, at most three bucket orders can be generated when we adopt EXTORDER to extend a bucket order by a new column (refer to Theorem 5.4). Thus, the EXTORDER procedure is bounded by $O(n)$ where n is the number of columns in the input matrix.

Theorem 5.4. *Given a bucket order $\tau = [Q^{(1)} \succ \cdots \succ Q^{(r)}]$ supported by row g, at most three bucket orders can be generated if τ is extended by a new column c.*

Algorithm 4: EXHROWEXP

Input: A GeBOPSM $(P, Q : \tau_Q)$, d_{max}, g_{min}, μ^*

1. **for** $g_i \in G - P$ **do**
2. $o_i = \text{DP-PICKBEST}(\tau_Q, g_i, d_{max}, g_{min})$;
3. **if** $d_{LCS}(\tau_Q, o_i) \geq \mu^*$ **then**
4. $P = P \cup \{g_i\}$; Add o_i to $O(P, Q)$;
5. **end**
6. **end**

Proof Sketch: If the value $M(g, c)$ falls in the range of a bucket, say $Q^{(i)}$, in τ, the only way that τ can be extended with c is to add c to bucket $Q^{(i)}$. If $M(g, c)$ falls in the gap between two adjacent buckets, say $Q^{(i)}$ and $Q^{(i+1)}$, we can extend τ by either adding c to $Q^{(i)}$, or adding c to $Q^{(i+1)}$, or creating a new bucket $\{c\}$ and inserting it between $Q^{(i)}$ and $Q^{(i+1)}$.

□

The row-wise expansion algorithm EXHROWEXP takes as input the current GeBOPSM $(P, Q : \tau_Q)$, the thresholds d_{max} and g_{min}, and the current smallest LCS similarity μ^* passed from the last call of GRDYCOLEXP. For every remaining row g_i in $(G - P)$, we pick a bucket order o_i among all the bucket orders supported by g_i, such that the LCS similarity between o_i and the backbone order τ_Q is maximized (Line 2). A *Dynamic Programming* (DP) based method DP-PICKBEST is adopted to efficiently find o_i and compute the LCS similarity. If the LCS similarity between τ_Q and o_i is no smaller than μ^*, we expand the current GeBOPSM by g_i and add o_i to $O(P, Q)$ (Lines 3 - 5).

DP-PICKBEST. Given a row g, the backbone order $\tau_Q = [Q^{(1)} \succ \cdots \succ Q^{(r)}]$ of a GeBOPSM, DP-PICKBEST checks whether g μ^*-supports τ_Q or not. We first sort the items (i.e., columns) in every bucket $Q^{(i)}$ according to the values of g under the corresponding columns. Then, the DP technique is adopted to find a bucket order that is supported by g, say τ_g, and that maximizes $|LCS(\tau_g, \tau_Q)|$. The underlying idea of DP function is that, for a particular item t in some $Q^{(i)}$ of τ_Q, the length of $LCS(\tau_g, \tau_Q)$ that ends with t is equal to the larger of the following two values. One is the length of $LCS(\tau_g, Q^{(i)})$ that ends with t. Another is the length of $LCS(\tau_g, \tau_Q)$ that ends with some other item in the bucket before $Q^{(i)}$, plus the number of items in $Q^{(i)}$ that can be appended.

5.3.2 Backbone Order Generation

When a GeBOPSM $(P, Q : \tau_Q)$ is expanded by a new column, its backbone order τ_Q needs to be updated accordingly. We adopt a median-rank based method called MEDBACKBONE to generate the backbone order of a GeBOPSM pattern. MEDBACKBONE takes as input all the bucket orders extended from all the rows in the GeBOPSM. For every bucket order, ranks are assigned to the items such that items in the first bucket have rank 1, items in the second bucket have rank 2, and so on. For each item, its rank profile is the set of ranks it gets in all the bucket orders. Then, items

Algorithm 5: SEEDGROWTH-COL

Input: A set U of BOPSMs, α, d_{max} and g_{min}
Output: A set V of GeBOPSMs

1. **for** $(P, Q : \tau_Q) \in U$ **do**
2. **if** (P, Q) *is a submatrix of a GeBOPSM in* V **then**
3. discard (P, Q);
4. **while** GRDYCOLEXP$((P, Q : \tau_Q), \alpha, d_{max}, g_{min}, \mu^*) == succeed$ **do**
5. continue;
6. **end**
7. ExhRowExp$((P, Q : \tau_Q), d_{max}, g_{min}, \mu^*)$;
8. Add $(P, Q : \tau_Q)$ to V;
9. **end**

with the same median rank form a bucket, and the buckets are sorted in the increasing order in terms of the median ranks that items in the buckets have. The generated bucket order is the backbone order of the GeBOPSM pattern.

5.3.3 *The* SEEDGROWTH *Algorithm*

Combining the GRDYCOLEXP and EXHROWEXP procedures in different order leads to different pattern growing strategies. Here, we adopt the *column-centric strategy* and the *row-centric strategy*.

The basic idea of the column-centric strategy is that the seed BOPSM (P, Q) is repetitively expanded column-wise by invoking GRDYCOLEXP until no more columns can be chosen for expansion. Then, one round of row-wise expansion is conducted by invoking EXHROWEXP as illustrated as follows.

$$(P, Q) \xrightarrow{\text{GRDYCOLEXP}} (P, Q \cup \{t_1\}) \xrightarrow{\text{GRDYCOLEXP}} \cdots$$
$$(P, Q \cup \Delta Q) \xrightarrow{\text{EXHROWEXP}} (P \cup \Delta P, Q \cup \Delta Q)$$

We implement the column-centric strategy as shown in Algorithm 5. First, we carry out an *early pruning* step (Lines 2-3) to check if a seed BOPSM is a submatirx of some mined GeBOPSMs. If it is the case, the seed BOPSM is discarded. Otherwise, SEEDGROWTH-COL starts the greedy column-wise expansion on the BOPSM repetitively until no more columns can be chosen (Lines 4-6). Then, a round of row-wise expansion is carried out by invoking EXHROWEXP (Line 7).

We also implement the row-centric pattern growing strategy. The basic idea is that, whenever the current pattern is expanded by a new column, the EXHROWEXP procedure is then invoked to further expand the current pattern with eligible rows. The row-centric strategy can be illustrated as follows.

$$(P, Q) \xrightarrow{\text{GRDYCOLEXP}} (P, Q \cup \{t_1\}) \xrightarrow{\text{EXHROWEXP}} (P \cup \Delta P_1,$$
$$Q \cup \{t_1\}) \xrightarrow{\text{GRDYCOLEXP}} \cdots \xrightarrow{\text{EXHROWEXP}} (P \cup \Delta P, Q \cup \Delta Q).$$

Using the row-centric strategy, another version of the SEEDGROWTH algorithm called SEEDGROWTH-ROW can be straightforwardly developed as follows: SEEDGROWTH-ROW first conducts the early pruning step on the seed BOPSM similar to SEEDGROWTH-COL. Then, it expands the seed BOPSM by taking GRDYCOLEXP and EXHROWEXP alternately until no more expansion can be conducted. We omit the details of SEEDGROWTH-ROW due to space limit.

6 EXPERIMENTS

In this section, we study the performance of the BOPSM model, the GeBOPSM model, and their mining methods APRIBOPSM and SEEDGROWTH on both synthetic datasets and a real biological dataset. We compare our algorithms with the state-of-the-art mining methods of other relaxed models, which include *OPC-Tree*, *AOPC* and *OPSM-Growth* as follows.

- *OPC-Tree* is a tree-based OPSM mining method [11], which exhaustively mines OPSM patterns that satisfy some size thresholds.
- The *AOPC* method [20] mines AOPC patterns, which takes a set of OPSM patterns as input, and merges them into AOPCs in a greedy way until no more valid AOPCs can be generated.
- The *OPSM-Growth* method [5] takes seed OPSMs as input and expands them into maximal ROPSM patterns.

All the above algorithms are implemented using C++, and all the experiments are conducted on a Macbook Pro with 2.53GHZ CPU and 4G memory.

6.1 Data Preparation

The synthetic datasets that are used to study the efficiency of the APRIBOPSM algorithm are generated as follows. We generate r-by-c matrices and vary the number of rows r to be $\{200, 400, 600, 800, 1000\}$ and the number of columns c to be $\{10, 15, 20, 25, 30\}$. The entry values are chosen within the range of $[1, 10]$. We implant 10 overlapping OPSMs along the diagonal, each with $15\% * r$ rows and $40\% * c$ columns in a generated matrix. Finally, we reorder the rows and columns to get the input matrix.

The real dataset we use is the cell cycle data of the yeast *Saccharomyces cerevisiae* [17][1]. It contains the transcription levels of 4000 genes under 24 experiment conditions, which are the arrest of a cdc15 temperature-sensitive mutant. The tool Gene Ontology Term Finder[2] is taken to validate the biological significance of the mined (BOPSM, GeBOPSM, OPSM, AOPC, or ROPSM) patterns. It computes the p-values between the mined patterns and known gene categories. A smaller p-value indicates a stronger association between the pattern and the category. Given a pattern, we count the number of categories with which it strongly associates. We take the commonly used p-value threshold, i.e., 1.0×10^{-9}, for determining

1. http://genome-www.stanford.edu/cellcycle/
2. http://search.cpan.org/dist/GO-TermFinder/

strong association as adopted in [20], [5]. In short, a pattern is said to be strongly associated with a known gene category if its p-value is less than 1.0×10^{-9}.

6.2 Evaluation Measures

We study the efficiency of different mining methods by comparing their execution time. Since different methods generate different number of patterns defined by their underlying model, we compare both the total running time (or simply the TR-time) and the average execution time for finding a single pattern (or simply the SP-time) in the evaluation.

Through the experiments on the real cell cycle datasets, we study the biological significance of mined patterns generated by different methods. For each method, we count for each generated pattern the number of known gene categories it strongly associates with. Then, we count the number and the fraction of generated patterns that are strongly associated with at least *a specific number of gene categories*, which we call *a significance level*. We then compare the number and the fraction of patterns that reach various significance levels generated by all the methods.

6.3 BOPSM and APRIBOPSM

6.3.1 Scalability

We study the scalability of the APRIBOPSM algorithm with respect to the matrix size and compare it with *OPC-Tree* using synthetic datasets. As *OPC-Tree* can only mine OPSM patterns, we set the thresholds d_{max} and g_{min} to be both 0 for fair comparison, which means that APRIBOPSM also mines OPSM patterns. There are still a few exceptional cases. For example, if the entry values of a row under several columns are exactly the same, APRIBOPSM considers this row supporting a bucket order. However, *OPC-Tree* considers the row supporting a linear order, which is a linearization of the bucket order. Thus, the number of OPSMs mined by APRIBOPSM and the number of OPSMs mined by *OPC-Tree* are not exactly the same.

First, we fix the number of columns c to be 15, and vary the number of rows r from 200 to 1000. From each r-by-c matrix, we mine the OPSM patterns that contain at least $10\% * r$ rows and $40\% * c$ columns. Figures 3(a) and 3(b) show the TR-time and the SP-time of APRIBOPSM and *OPC-Tree*. Clearly, our APRIBOPSM algorithm is significantly more efficient than *OPC-Tree*. As the number of rows increases, both the TR-time and the SP-time of APRIBOPSM increase slowly and linearly, while the TR-time and the SP-time of *OPC-Tree* exhibit a sharp increase. For example, when the number of rows increases 5 times (from 200 to 1000), the TR-time of APRIBOPSM also increases about 5 times (from 3.19 secs to 14.65 secs). In comparison, the TR-time of *OPC-Tree* increases about 20 times (from 18.7 secs to 371.5 secs). Similar conclusions can be drawn when analyzing the SP-time. The fact that APRIBOPSM is more efficient than *OPC-Tree* can be explained as follows. APRIBOPSM generates candidate bucket orders in a breadth-first way, which controls better the growth of the number of candidates during the candidate generation step. Moreover, the adoption of the *BPTree* makes this step quite efficient. In contrast, the depth-first *OPC-Tree* method generates excessively large number of candidates before pruning those ineligible ones, which is rather time-consuming.

We then study how the number of columns influences the performance of the algorithms. We fix r to be 200, vary c from 10 to 30, and mine from each r-by-c matrix the OPSM patterns that contain at least $10\% * r$ rows and $40\% * c$ columns. Figures 3(c) and 3(d) show that APRIBOPSM also outperforms *OPC-Tree* significantly.

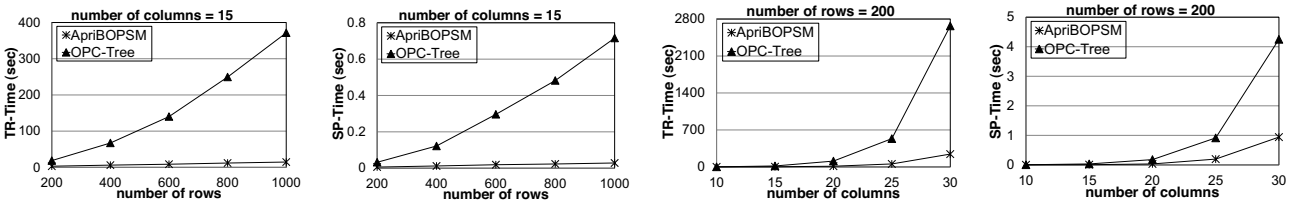
On the other hand, the performance of both APRIBOPSM and *OPC-Tree* is more influenced by column increase than by row increase. The reason may be that, for both methods, an increase of the number of columns theoretically leads to the exponential increase of possible candidates. However, APRIBOPSM better controls the growth of the number of generated candidates during the mining process, and thus it enjoys a relatively better performance as a result.

6.3.2 Influence of Thresholds g_{min} and d_{max}

Next, we use the real dataset to study the impact of the thresholds g_{min} and d_{max} on the execution time of APRIBOPSM and the quality of the mined BOPSM patterns. As discussed in Section 4, we can set for each row in the input matrix its own g_{min} and d_{max} thresholds concerning different scalings of the rows. In this set of experiments, for each row, we vary its g_{min} threshold from 0 to 0.5 times the average gap and vary its d_{max} threshold from 0 to 2 times the average gap. Note that the average gap of different rows may be different, and also g_{min} being equal to 0 actually means that g_{min} is set to be some number that is smaller than the smallest gap. We mine the BOPSM patterns with at least 80 rows and 6 columns.

Figures 4(a) and 4(b) show the TR-time and the SP-time with respect to g_{min}. The value along the x-axis, say 0.2, means that g_{min} equals 0.2 times the average gap. The series in the legend, say $d_{max} = 2$, means that d_{max} equals 2 times the average gap. As g_{min} gets larger, more patterns are pruned, and thus the TR-time decreases. However, the SP-time cost for finding a single pattern increases.

Figures 4(c) and 4(d) show the TR-time and the SP-time with respect to d_{max}, and the meaning of the series name in the legend and the x-axis is similarly interpreted. As d_{max} gets larger, more patterns are mined, and thus the TR-time increases. However, when d_{max} is larger than 1.5 times the average gap, the number of patterns mined increases significantly, which makes the SP-time decreases.



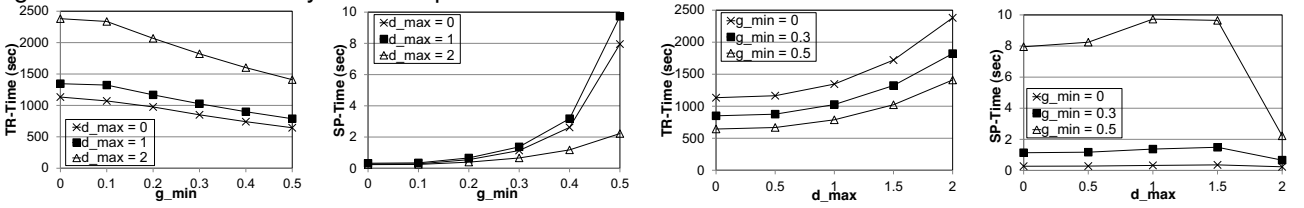
(a) TR-time w.r.t. the number of rows

(b) SP-time w.r.t. the number of rows

(c) TR-time w.r.t. the number of columns

(d) SP-time w.r.t. the number of columns

Fig. 3. BOPSM - Scalability With Respect To the Size of Matrix



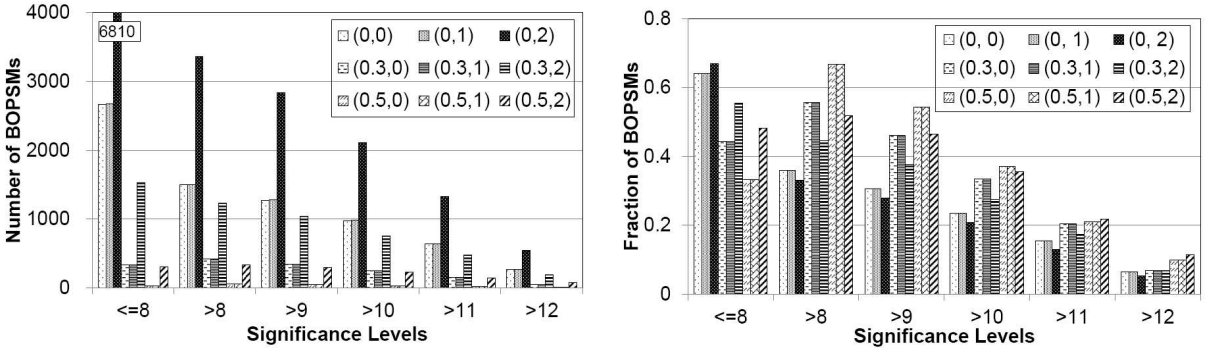
(a) TR-time w.r.t. g_{min}

(b) SP-time w.r.t. g_{min}

(c) TR-time w.r.t. d_{max}

(d) SP-time w.r.t. d_{max}

Fig. 4. BOPSM - Execution Time With Respect To g_{min} and d_{max}



(a) Number of significant BOPSMS

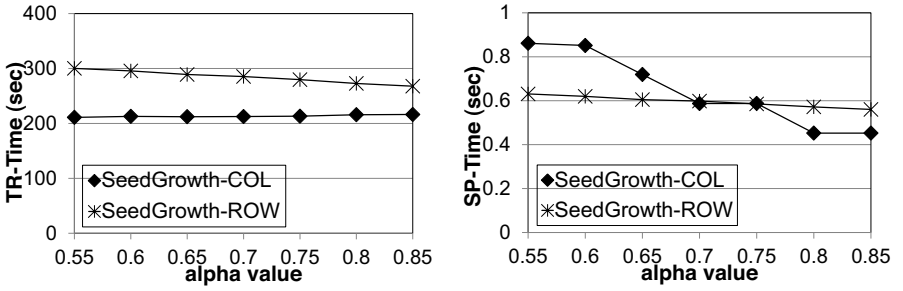
(b) Fraction of significant BOPSMs

Fig. 5. BOPSM - Quality of BOPSMs With Respect To g_{min} and d_{max}

Figures 5(a) and 5(b) show the number and fraction of BOPSMs that reach different significant levels under various g_{min} and d_{max} combinations. The series name, say $(0.5, 1)$, means that g_{min} equals 0.5 average gap and d_{max} equals 1 average gap. When g_{min} is large, smaller number of patterns are mined. However, the quality of the mined patterns increases apparently. For example, when g_{min} equals 0, less than 36% BOPSMs reach the significance level " > 8", however, when g_{min} equals 0.5 average gap, more than 45% BOPSMs reach the significance level " > 8". Consider the influence of d_{max}. Generally, when d_{max} gets larger, the number of mined BOPSMs increases. When g_{min} is small and d_{max} gets larger than 1 average gap, the quality of the BOPSMs decreases, which indicates that the mined patterns may be too noisy in this case. However, if we set g_{min} to be large, say, 0.5 average gap, and increase the d_{max} value, the fraction of BOPSMs that reach significance levels " > 11" and " > 12" increases. This interesting finding implies that the quality of the patterns can be maintained by using an appropriate combination of these two thresholds.

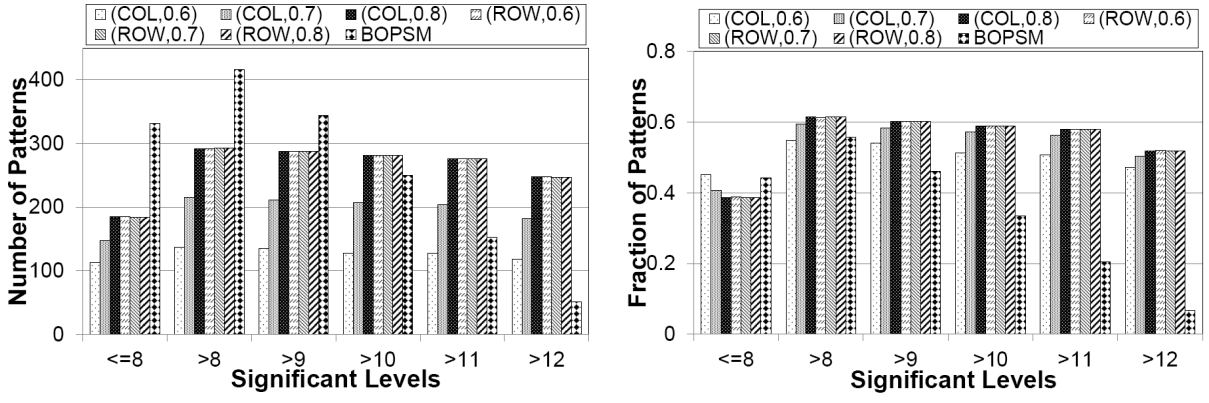
6.4 GeBOPSM and SEEDGROWTH

We study the effectiveness of the GeBOPSM model and the performance of the SEEDGROWTH algorithm using the real dataset. By adopting different pattern growing strategies, we have SEEDGROWTH-COL corresponding to the column-centric growing strategy and SEEDGROWTH-ROW corresponding to the row-centric growing strategy. We adopt the APRIBOPSM alogrithm to mine the seed BOPSMs with at least 80 rows and 6 columns and with the thresholds g_{min} and d_{max} set to be 0.3 average gap and 0. APRIBOPSM mines 747 BOPSMs, which are taken as the input for both SEEDGROWTH-COL and SEEDGROWTH-ROW.



(a) TR-time w.r.t. α (b) SP-time w.r.t. α

Fig. 6. GeBOPSM - Execution Time of Mined GeBOPSMs ($g_{min} = 0.3, d_{max} = 0$)



(a) Number of significant GeBOPSMs (b) Fraction of significant GeBOPSMs

Fig. 7. GeBOPSM - Quality of Mined GeBOPSMs ($g_{min} = 0.3, d_{max} = 0$)

6.4.1 Efficiency

We first study the impact of similarity threshold α on SEEDGROWTH-COL and SEEDGROWTH-ROW. Figures 6(a) and 6(b) show the TR-time and the SP-time of these two algorithms against α.

From Figure 6(a), we can see that, as the similarity threshold α increases, the TR-time of SEEDGROWTH-COL remains roughly the same, while the TR-time of SEEDGROWTH-ROW shows a slight decrease. Compared to SEEDGROWTH-ROW, SEEDGROWTH-COL costs less time. This indicates that, when the column-centric strategy is adopted, the seed BOPSMs are more likely to grow into larger GeBOPSMs, which may prune more seed BOPSMs during the pruning step. It can be checked that in Figure 6(b) the SP-time of both SEEDGROWTH-COL and SEEDGROWTH-ROW decreases as α increases. It is reasonable, since less noise is allowed in the mined patterns for larger α, then less time is needed for growing a pattern as the mining result. Another interesting finding in Figure 6(b) is that SEEDGROWTH-COL costs less SP-time when α is large, while SEEDGROWTH-ROW costs less SP-time when α is small. This indicates that SEEDGROWTH-COL likely generates larger patterns especially when α is small, and thus more rounds of column-wise expansion are needed, which is relatively more time consuming.

6.4.2 Biological Significance

Figures 7(a) and 7(b) show the biological significance of mined GeBOPSMs and the seed BOPSMs, where the respective series names represent the combinations of the versions of SEEDGROWTH and α values. For example, (COL, 0.7) means that SEEDGROWTH-COL is used with $\alpha = 0.7$. Considering the number of significant patterns in Figure 7(a), we can see that, at the lower significance levels like " > 8" and " > 9", the number of seed BOPSMs is obviously larger than the number of GeBOPSMs mined by SEEDGROWTH under all settings. However, at the higher significance levels like " > 10", " > 11", or " > 12", the number of GeBOPSMs becomes increasingly larger than that of the seed BOPSMs. This indicates that the adoption of the GeBOPSM model largely improves the quality of the mined patterns. The statistics about the fraction of significant patterns which are shown in Figure 7(b) more clearly support this claim. For those levels that are higher than " > 8", the fraction of GeBOPSMs mined by SEEDGROWTH under every setting is larger than that of the seed BOPSMs. Specifically, only 20% seed BOPSMs reach the significance level " > 11", while at least 56% GeBOPSMs reach this level.

In the above set of experiments, we keep g_{min} to be 0.3 average gap and d_{max} to be 0, both of which are the same as that we adopted to mine the seed BOPSMs. If we set g_{min} to be 0.3 average gap and d_{max} to be 2 average gaps, the qualities of the mined GeBOPSMs are further improved. Specifically, the fractions of GeBOPSMs that reach the significant levels higher than "> 10" all increase by 4% to 7%. It further supports our findings in previous experiments that using the appropriate settings of the difference and gap thresholds is able to improve the quality of mined patterns.

6.5 Comparisons with Related Methods

Finally, we compare the biological significance of five different types of patterns: GeBOPSM, BOPSM, ROPSM, AOPC, and OPSM. The GeBOPSM mining method takes the BOPSM patterns as input, while the ROPSM and AOPC mining methods take the OPSM patterns as input. To achieve fair comparison, we adopt APRIBOPSM to mine BOPSM patterns by setting g_{min} and d_{max} to be 0.3 average gap and 0. The resultant 747 BOPSMs are also OPSMs, which are taken as the input for the GeBOPSM, ROPSM, and AOPC mining methods. The statistical results of all the methods are presented in Table 3. The detailed settings for the mining methods are given as follows.

1) **GeBOPSM** - We adopt SEEDGROWTH-COL to mine GeBOPSM patterns with α set to be 0.7. The thresholds g_{min} and d_{max} are set to be 0.3 average gap and 2 average gaps.
2) **ROPSM** - Similar to SEEDGROWTH, *OPSM-Growth* [5] also has the column-centric version and the row-centric version. We studied both versions by using the best similarity threshold according to the quality of the mined ROPSMs.
3) **AOPC** - We implemented the AOPC mining method in [20], and conducted the experiments by varying the similarity requirement and the number of initial groups. The AOPCs with the best quality are then chosen for comparison.
4) **BOPSM** - The APRIBOPSM algorithm is adopted to mine BOPSM patterns with at least 80 rows and 6 columns. We set the thresholds g_{min} and d_{max} to be 0.5 average gap and 1 average gap.
5) **OPSM** - The OPSM patterns are the input of the GeBOPSM, ROPSM and AOPC mining methods, and the statistics are listed in the last column.

In Table 3, we present the best results (highlighted in bold font) in terms of the fraction of patterns for all significance levels except "≤ 7". It can be verified that more than 80% of the BOPSM patterns reach the significance level "> 7", while other types are no more than 70% at the same level. However, for higher significance levels ("> 10", "> 11", and "> 12"), the fractions of GeBOPSMs, ROPSMs and AOPCs, which all adopt the similarity relaxation strategy, are larger than the fractions of both BOPSMs and OPSMs.

Among GeBOPSM, ROPSM and AOPC, the SP-time of (AOPC, 8, 0.6) and (ROPSM, COL, 0.7) are the shortest. However, for all significant levels higher than "> 7", the fractions of the patterns mined under these two settings are always smaller than the fractions of the patterns mined under the settings of (GeBOPSM, COL, 0.7) and (ROPSM, ROW, 0.6). Referring to (GeBOPSM, COL, 0.7) and (ROPSM, COL, 0.7), which use the same pattern growing strategy and similarity threshold, the SP-time cost for finding a GeBOPSM pattern is longer than the SP-time cost for finding an ROPSM pattern. However, the quality of the mined GeBOPSMs is apparently better than that of the ROPSMs. For example, the fraction of GeBOPSMs is 10% higher than that of the ROPSMs at the significance levels "> 11" and "> 12". The advantage is even more obvious when the absolute number of patterns are checked. When (GeBOPSM, COL, 0.7) is compared to the best ROPSM result, i.e., (ROPSM, ROW, 0.6), the quality of GeBOPSMs is still slightly better than that of ROPSMs, but the time for mining a GeBOPSM is only half of that for mining an ROPSM.

7 CONCLUSIONS

In this paper, we study the problem of mining relaxed OPSM patterns in order to discover significant biological associations in gene expression data.

We propose the BOPSM model that captures the biological fact that some correlated genes follow a consensus trend identified as a bucket order. The model requires that the rows in a BOPSM support a backbone bucket order. The structure of the backbone bucket order can be monitored by the intra-bucket difference and inter-bucket gap thresholds. We also develop a BOPSM mining method called APRIBOPSM, which makes use of an Apriori-based framework and adopts a novel *BucketPrefixTree* structure to mine BOPSM patterns. Experiments on both synthetic and real datasets confirm that, the BOPSM model facilitates much better the discovery of quality but noisy OPSM patterns than the strict OPSM model. Our mining method is also significantly more efficient than *OPC-Tree*.

We propose the GeBOPSM model, which allows that bucket orders are similar enough to the backbone bucket order in an OPSM pattern. The GeBOPSM model generalizes all the existing relaxed OPSM models. We also develop the GeBOPSM mining method SEEDGROWTH and propose two different pattern growing strategies, i.e., column-centric or row-centric, to grow seed BOPSMs into GeBOPSM patterns. Experimental studies show that the GeBOPSM model outperforms all the current relaxed OPSM models, and importantly, it leads to the discovery of more quality patterns in terms of both fraction and number.

TABLE 3
Comparison of GeBOPSM, ROPSM, AOPC, BOPSM, and OPSM

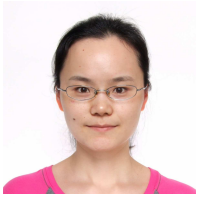
	(GeBOPSM,COL,0.7)	(ROPSM,ROW,0.6)	(ROPSM,COL,0.7)	(AOPC,8,0.6)	BOPSM	OPSM
Pattern number	459	134	252	294	81	747
TR-time	329.56 sec	209.64 sec	51.78 sec	55.66 sec	–	–
SP-Time	0.72 sec	1.56 sec	0.21 sec	0.19 sec	–	–
the number (fraction) of patterns that reach each significance level						
> 12	**270(58.8%)**	70(52.2%)	123(48.8%)	79(26.9%)	8(9.9%)	51(6.8%)
> 11	**292(63.6%)**	82(61.2%)	135(53.6%)	113(38.4%)	17(21.0%)	153(20.5%)
> 10	**293(63.8%)**	85(63.4%)	140(55.6%)	127(43.2%)	30(37.0%)	250(33.5%)
> 9	**299(65.1%)**	87(64.9%)	146(57.9%)	149(50.7%)	44(54.3%)	344(46.1%)
> 8	305(66.5%)	89(66.4%)	153(60.7%)	160(54.4%)	**54(66.7%)**	416(55.7%)
> 7	312(68.0%)	93(69.4%)	160(63.5%)	169(57.5%)	**66(81.5%)**	467(62.5%)
≤ 7	147(32.0%)	41(30.6%)	92(36.5%)	125(42.5%)	15(18.5%)	280(37.5%)

ACKNOWLEDGMENT

This work is partially supported by Hong Kong RGC GRF project grant 617610, Hong Kong RGC TBRS project grant T12/CUHK03/10, and China NSF Grant 60970043.

REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE '95*, pages 3–14, 1995.
[2] A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: the order-preserving submatrix problem. In *RECOMB '02*, pages 49–57, 2002.
[3] Y. Cheng and G. M. Church. Biclustering of expression data. In *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, pages 93–103, 2000.
[4] C. K. Chui, B. Kao, K. Y. Yip, and S. D. Lee. Mining order-preserving submatrices from data with repeated measurements. In *ICDM '08*, pages 133–142, 2008.
[5] Q. Fang, W. NG, and J. Feng. Discovering significant relaxed order-preserving submatrices. In *SIGKDD '10*, pages 433–442, 2010.
[6] B. J. Gao, O. L. Griffith, M. Ester, and S. J. M. Jones. Discovering significant opsm subspace clusters in massive gene expression data. In *SIGKDD '06*, pages 922–928, 2006.
[7] N. Gupta and S. Aggarwal. Mib: Using mutual information for biclustering gene expression data. *Pattern Recognition*, 2010.
[8] R. Gupta, N. Rao, and V. Kumar. Discovery of error-tolerant biclusters from noisy gene expression data. In *BIOKDD '10*, 2010.
[9] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1–58, 2009.
[10] G. Li, Q. Ma, H. Tang, A. Paterson, and Y. Xu. Qubic: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research*, 37(15), 2009.
[11] J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. In *ICDM '03*, 2003.
[12] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM TCBB*, 1(1):24–45, 2004.
[13] G. Pandey, G. Atluri, M. Steinbach, C. L. Myers, and V. Kumar. An association analysis approach to biclustering. In *SIGKDD '09*, pages 677–686, 2009.
[14] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, and et al. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE '01*, 2001.
[15] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, and et al. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE TKDE*, 16:1424–1440, 2004.
[16] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, and et al. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, 2006.
[17] P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9(12):3273–3297, December 1998.
[18] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT '96*, pages 3–17, 1996.
[19] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18:136–144, 2002.
[20] M. Zhang, W. Wang, and J. Liu. Mining approximate order preserving clusters in the presence of noise. In *ICDE '08*, pages 160–168, 2008.



Qiong Fang received her BSc and MPhil degrees in computer science from Huazhong University of Science and Technology (HUST). She is currently a PhD candidate in the Department of Computer Science and Engineering at the Hong Kong University of Sicence and Technology (HKUST). Her research interests are in the areas of data mining and bioinformatics.

Wilfred Ng received his MSc (Distinction) and PhD in Computer Science from the University of London. Currently he is an Associate Professor of Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST), where he is a member of the database research group. His research interests are in the areas of databases, data mining and information Systems, which include Web data management and XML searching. Further Information can be found at the following URL: http://www.cs.ust.hk/faculty/wilfred/index.html.

Jianlin Feng received his BSc, MSc, and PhD degrees in computer science from Huazhong University of Science and Technology. Currently he is a full professor in the School of Software at Sun Yat-Sen University (SYSU). His research interests are in the areas of data mining and database systems.

Yuliang Li is currently an undergraduate student in the Department of Computer Sicence and Engineering at the Hong Kong University of Science and Technology (HKUST). His research interests are in the areas of data mining and bioinformatics.