Mining Approximate Order Preserving Clusters in the Presence of Noise

Mengsheng Zhang[#1], Wei Wang[#2], Jinze Liu[*3]

[#]*Department of Computer Science, University of North Carolina at Chapel Hill*
Chapel Hill, NC 27599-3175 USA

[1]`mszhang@cs.unc.edu`
[2]`weiwang@cs.unc.edu`

[*]*Department of Computer Science, University of Kentucky*
Lexington, KY 40506-0046 USA

[3]`liuj@cs.uky.edu`

***Abstract*— Subspace clustering has attracted great attention due to its capability of finding salient patterns in high dimensional data. Order preserving subspace clusters have been proven to be important in high throughput gene expression analysis, since functionally related genes are often co-expressed under a set of experimental conditions. Such co-expression patterns can be represented by consistent orderings of attributes. Existing order preserving cluster models require all objects in a cluster have identical attribute order without deviation. However, real data are noisy due to measurement technology limitation and experimental variability which prohibits these strict models from revealing true clusters corrupted by noise. In this paper, we study the problem of revealing the order preserving clusters in the presence of noise. We propose a noise-tolerant model called approximate order preserving cluster (AOPC). Instead of requiring all objects in a cluster have identical attribute order, we require that (1) at least a certain fraction of the objects have identical attribute order; (2) other objects in the cluster may deviate from the consensus order by up to a certain fraction of attributes. We also propose an algorithm to mine AOPC. Experiments on gene expression data demonstrate the efficiency and effectiveness of our algorithm.**

I. Introduction

In recent years, the advent of high throughput data generation techniques have increased not only the number of objects collected in databases, but also the number of attributes describing these objects. The resultant datasets are often referred to as high dimensional. Clustering high dimensional data using traditional algorithms has suffered from the fact that many attributes may be irrelevant and can thus mask clusters located in some subspaces. Subspace clustering algorithms have recently been proposed to solve this problem. They search for clusters in subspaces formed by relevant attributes [1]. Among various subspace clustering models, one was designed to mine a set of objects which show identical attribute order, called *order preserving cluster* (OPC) [6]. We will give a formal description of this model in next section. This model originally attracts researchers' interests because of its important utility in gene expression data analysis. Based on the understanding of cellular processes, it is a general belief that some subsets of genes may be co-expressed under certain experimental conditions, but behave independently under other conditions. Finding such local expression patterns exhibited under relevant conditions is one important contribution of the OPC algorithm and may be the key to uncover significant but previously unknown genetic pathways.

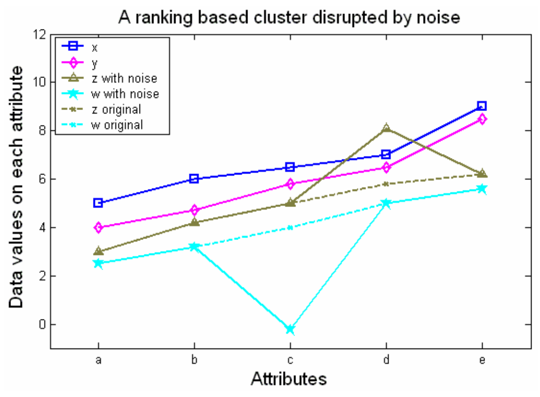

Fig. 1 An example illustrates how an OPC is disrupted by noise. Objects *w* and *z* are excluded from the cluster due to noise

However, noise is ubiquitous in real data due to technical errors, missing values and variable experimental conditions, etc. The underlying OPCs may be broken into small ones by noise and cannot be captured by any strict model due to their vulnerability to noise. Figure 1 shows an example. The dataset contains four objects $\{w, x, y, z\}$ with attributes $\{a, b, c, d, e\}$. Originally, all objects follow the same attribute order: their values on *a*, *b*, *c*, *d* and *e* are in increasing order. Thus $\{w, x, y, z\}$ is an OPC on $\{a, b, c, d, e\}$. However, after distributing some noise into this dataset, *w* and *z* deviate from their original attribute order. As a result, $\{w, x, y, z\}$ is no longer an OPC on $\{a, b, c, d, e\}$. Instead, it is broken into several smaller ones with overlap, such as $\{x, y, z\}$ on $\{a, b, c, d\}$, $\{w, x, y\}$ on $\{a, b, d, e\}$ and $\{x, y\}$ on $\{a, b, c, d, e\}$. From this example we can see that the true cluster cannot be captured by the strict model in the presence of noise.

Mining subspace OPCs in the presence of noise is very challenging for the following reasons. First, the search space is often huge due to the curse of dimensionality. For a dataset with n attributes, there are totally 2^n candidate subspaces. For

OPC mining, the complexity is much higher, since in addition to identifying subspaces, we also need to distinguish different orders. For a subspace of m attributes, there are totally $m!$ attribute orders. So given a dataset with n attributes, the total number of possible orders for all subspaces is:

$$\sum_{m=1}^{n} \binom{n}{m} \times m! \quad (1)$$

For each attribute order, there could be a potential OPC associated with it. This means even for a dataset with 10 attributes, there are $O(10^6)$ orders to search. Second, when taking noise into account, the problem becomes even harder. Ambiguity is brought into the problem by noise. Objects in the same cluster may not have identical attribute order anymore. Thus given a cluster candidate, how to identify its consensus attribute order becomes a new challenge which was not an issue in previous models. Third, some good properties such as the anti-monotonicity do not hold anymore when tolerating noise. All of above facts pose significant challenges for OPC mining in the presence of noise.

To our best knowledge, no previous work has been done on the problem of subspace OPC mining with noise tolerance. In this paper, we study this problem and propose a new model. Experimental results demonstrate that our new model can capture OPCs contaminated by noise and thus is more robust to noise than the previous strict model. We also propose an algorithm to mine clusters under the new model. Although we deal with a much more challenging problem, our algorithm can discover more significant clusters that cannot be found by the previous strict model in an efficient way.

The remainder of this paper is organized as follows. Section II is the preliminary section. It introduces the notations and terminologies we use throughout the paper. Section III gives a brief review of related work. In Section IV, we propose our new model and some experimental evidence to demonstrate its noise tolerance capability. Section V presents the algorithm, where we first propose a basic mining algorithm then followed by the discussion of several optimization techniques. Section VI shows the experimental results. We conduct a series of experiments to evaluate both the efficiency and effectiveness of our algorithm. We conclude the paper in Section VII.

II. Preliminaries

In this section, we discuss the terminologies, notations and assumptions of this paper. We also formally define the OPC mining problem here. First, some notations we use are listed as below:

D	A set of objects
A	The set of all attributes of objects in D
(C, T)	A subset of input dataset, $C \subseteq D$, $T \subseteq A$
$x, y..$	Individual object in D
$a, b..$	Individual attribute in A
d_{xa}	Value of object x on attribute a

The (attribute) *order* of an object x on a subset of attributes T is a permutation of the attributes in T induced by the values of x on these attributes. The order of x on T, denoted by o_{xT}, is $o_{xT} = abc\ldots$ if and only if:

$$d_{xa} < d_{xb} < d_{xc} < \ldots \quad (2)$$

If the order of x on T is o_{xT}, we also say x *follows* o_{xT} or o_{xT} is *supported* by x. We may omit the subscripts for convenience if there is no ambiguity. For example, x and y are two objects with five attributes $\{a, b, c, d, e\}$ whose values are shown in TABLE I. In this example x and y follow orders *bcead* and *daecb* respectively. A *sub* (or *super*) sequence of o_{xT} is referred to as a *sub* (or *super*) *order* of o_{xT}. A subset of the dataset, (C, T) $(C \subseteq D, T \subseteq A)$, is an *order preserving cluster* (OPC) [6] if and only if all objects in C follow the same order on T. We may also require a minimum size for a cluster ($|C| \geq s_{min}$, $|T| \geq l_{min}$). Only clusters satisfy $|C| \geq s_{min}$ and $|T| \geq l_{min}$ are *valid*. In the remainder of this paper when we refer to an OPC, we mean a *maximal* one[1] unless otherwise specified. The *OPC mining problem* of a given dataset D is to find all valid OPCs in D. Since no noise is allowed, OPC model is a strict model.

TABLE I
Values of x and y on Each Attribute and Their Orders

	a	***b***	***c***	***d***	***e***	**order**
x	4	1	2	5	3	*bcead*
y	7	10	9	5	8	*daecb*

III. Related Work

Clustering is the process of grouping *similar* objects. Various similarity measures have been employed, some are based on distances and others are based on patterns. OPC model belongs to the second category. In this section we review previous work on pattern based clustering. Since none of them considers the presence of noise, we call them *strict models*. A common characteristic of the strict models is that all objects in a cluster must follow the same pattern. Beside the strict models, we also review the work of approximate frequent itemset mining, since its idea of handling noise is related to our problem.

The earliest work on pattern based clustering is the bi-cluster model proposed by Cheng *et al.* in [3]. This model tries to measure the coherence between genes and the experiment conditions in a sub-matrix of a DNA array. Later, Wang *et al.* proposes δ-pCluster model [4] which aims to discover clusters of objects show shifting or scaling patterns. With powerful pruning strategies, δ-pClusters can be mined efficiently. However, this algorithm can group objects exhibiting either pure shifting or pure scaling pattern, but not both at the same time. Xu *et al.* proposes another model called Reg-Cluster [5] to relax this limitation. Through a linear transformation, this model is able to capture the shifting and scaling patterns simultaneously. But the problem is: for most real world applications, requiring exact shifting or scaling is

[1] That is, the OPC is not the subset of any other OPC.

too restrictive. In order to include more diverse patterns, we often need to lower the required pattern significance. This, in turn, can result in undesirable inconsistency inside a cluster.

The concept of OPC was first proposed by Ben-Dor *et al.* in [2] where they called it *order preserving sub-matrix* (OPSM). Each OPSM represents a subset of genes identically ordered among a subset of experiment conditions in a gene Micro-array dataset. Since this problem is NP-hard, they proposed a probabilistic model to mine an OPSM from a random matrix. The local patterns found by this algorithm seem to be significant. A drawback of this algorithm is that only one cluster can be found at a time and the result is very sensitive to input parameters and initial seeds. To find multiple OPSMs at the same time, Liu *et al*. proposes a deterministic algorithm to mine all OPCs in [6]. They develop an efficient pruning strategy and an auxiliary data structure called OPC-tree, this algorithm searches the full order space and thus can find all orders exhibited by a subset of objects along a subset of attributes. The OPC model is more flexible than those models only capturing specific patterns such as shifting and scaling, thus can be applied more widely. However due to the noisy nature of real data, it still fail to discover some significant clusters, since it requires all objects in a cluster have identical order and thus excludes those objects originally in the cluster but contaminated by noise. We will show that the noise tolerance capability of OPC models is very weak in the next section.

The task of frequent itemset mining is to mine a sub-matrix of '1's containing a sufficiently large set of rows (transactions) in a binary matrix representation of the input dataset. This problem also suffers from the presence of noise which may corrupt true frequent itemsets. Liu *et al.* proposes a noise tolerant model for this problem called *approximate frequent itemset* (AFI) in [7]. AFI criteria place restrictions on the fraction of noise in both the rows and columns of a sub-matrix which ensures a relatively uniform distribution of noise in any discovered patterns. AFI is proven to be effective in revealing significant underlying frequent itemsets. However, it only deals with binary data. For continuous data, it is much more difficult to discover noise-tolerant clusters. In this paper, we study noise tolerance in continuous data.

IV. MODEL

A. *Approximate Order Preserving Cluster*

Due to the noisy nature of real data, it is often too optimistic to expect all objects in a cluster to have the same order. Co-expression patterns in gene expression data is such an example. So in order to find more significant clusters, a more flexible model which can tolerate noise is needed. In this section, we propose a new model called *approximate order preserving cluster* (AOPC). The general idea is that the members of an AOPC should follow similar (not necessarily identical) orders. At the same time, there should be enough members supporting an order as the *consensus* order of the cluster. The novelty of this model is that it allows relaxation in a systematic way. Instead of requiring all objects have an identical order, it allows a group of objects with similar orders to form a cluster. The formal definition of this model is in DEFINITION 4.1. In this definition, *LCS*(*a*, *b*) denotes the *longest common subsequence* of two sequences *a* and *b* [8]. Also, we use $|\cdot|$ to denote the size (or length) of a set (or sequence). δ_c and δ_s are two input parameters of AOPC model to control the allowed noise level, both are between 0 and 1. We will provide some guidance on how to choose them in real world applications later.

DEFINITION 4.1 *Given a dataset D and its attribute set A. Let (C, T) be a subset of the dataset where $C \subseteq D$ and $T \subseteq A$. If o is an order of attributes in T, then C is an* ***approximate order preserving cluster*** *with order o if and only if it satisfies the following two criteria:*

1. *(consistency criterion) For each object x in C, $|LCS(o_{xT}, o)| \geq |o| \times \delta_c$, where $0 < \delta_c \leqslant 1$.*
2. *(supporting criterion) There exist at least $|C| \times \delta_s$ objects in C which support o, where $0 < \delta_s \leqslant 1$.*

From the above definition, we know that the OPC model is actually a special case of AOPC where $\delta_s = 1$. Thus an OPC is an AOPC as well. Moreover, we define the *core* of an AOPC in DEFINITION 4.2 which is another important concept for our algorithm.

DEFINITION 4.2 *Let (C, T) be an AOPC with order o, then its* ***core*** *is the subset of C consisting of all objects in O which support o.*

B. *Robustness to Noise*

In this section, we present some experimental results to demonstrate that the AOPC model is more robust to noise than the OPC model.

The objective of the experiment is to compare the noise tolerant capability of the OPC model and the AOPC model. The experiment is designed in the following way. First, we generate an OPC where all objects in it follow an identical order. The value of each data entry in this OPC satisfies a normal distribution with mean zero and variance one. Second, we add some noise to each data entry. The noise also satisfies a normal distribution with mean zero and a controlled variance σ. We set σ to zero initially and increase it gradually by 10^{-7} at each step. With each σ value, we test the satisfiability of the resulting data for both OPC and AOPC models. At the beginning, since there is no noise in the dataset, it satisfies both models. As σ increases, the resulting data becomes messier. There exists some σ from which the data no longer satisfies the model, we call this value *turning point*. We want to find turning points for OPC model and AOPC model respectively and compare their magnitude difference. In our experiment, we totally generated 1000 clusters with various sizes. The parameters δ_s and δ_c in AOPC model are set to be 0.2 and 0.6 respectively. For each of the 1000 test cases, we record its turning points for OPC model and AOPC model respectively. We plot histograms to compare the turning point distribution under each model, as shown in Figure 2. To make the figure more readable, we use a logarithmic scale $\lg(\sigma)$ for the turning points (x-axis). In Figure 2, the blue histogram shows the turning point distribution under OPC model while the red one is under AOPC model. From the result we can see that most turning points of OPC model are around 10^{-5} while the turning points of AOPC model are mostly around 10^{-2}. So the turning points of OPC model are generally much smaller than that of AOPC model. Small turning point indicates poor robustness to noise. This experiment demonstrates that OPC model fails to recognize the clusters when very small perturbations are added while AOPC model can still discover the original clusters in the presence of much higher level noise. The big gap between blue and red distributions shows that AOPC model is much more robust to noise than OPC model, which is the key to discover significant clusters in real data.

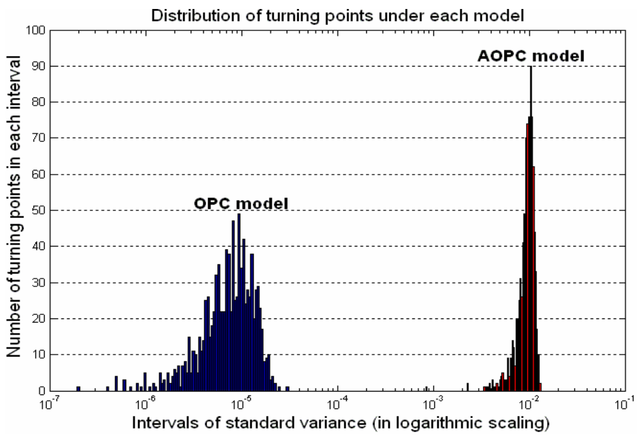

Fig. 2 Blue and red histograms are the turning point distributions of the OPC model and AOPC model respectively. The histograms are computed from 1000 test cases. Note the x-axis is in lg(σ)

V. Mining AOPC

In this section, we propose an algorithm to mine AOPCs in a given dataset. Our algorithm can be divided into two phases. We find all valid OPCs in the given dataset in Phase 1. In Phase 2, we mine AOPCs from the result of Phase 1. The general idea of this algorithm is inspired by the observation we made in the introduction. As shown in Figure 1, the presence of noise breaks original clusters into smaller ones with overlap. So a natural thought is to reverse this process, i.e. first find all small OPCs and then merge them to recover the true clusters.

A. Mine OPCs

Given a dataset *D*, user-specified parameters s_{min} and l_{min}, Phase 1 finds all valid OPCs, i.e. with size at least s_{min} and order length at least l_{min}. We modify the original OPC mining algorithm in [6]. This algorithm exhaustively enumerates all OPCs from lower-dimensional space to higher-dimensional space by adding one attribute at a time. It traverses the search space in a depth-first order. The anti-monotonic property is applied to prune the candidate subspaces that do not contain any valid clusters.

We made some modifications to the original algorithm to make it work well with Phase 2. The major changes we made are as below. First, the original algorithm has a pre-processing phase which groups attributes with similar values together. This process not only introduces extra computation, but also has the risk of missing some OPCs. In our algorithm, in order to find more OPCs as a good foundation for Phase 2, we remove this pre-processing phase. Second, the original algorithm returns a set of OPCs without any order. However, as we will explain later, to organize the OPCs in a way such that similar OPCs are adjacent to each other is important for the efficiency of Phase 2. Since the OPC algorithm traverses the order space in depth-first manner, the OPCs generated in consecutive steps are likely to have similar orders thus similar to each other. Figure 3 illustrates this with an example. It shows the search process of a set of three attributes $\{a, b, c\}$. By depth-first searching manner, similar attribute orders such as *ab*, *abc*, *ac* and *acb* are traversed in consecutive steps. When the total number of attributes increases, this property is more prominent. In order to take advantage of the temporal locality of similar OPCs generated during the traversal of the search space, we use a FIFO queue to store the OPCs in the order of which they are generated. Due to space limitation, we cannot cover every detail here. For more details of the OPC algorithm, please refer to [6].

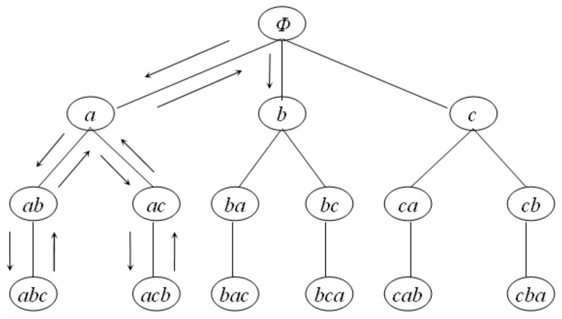

Fig. 3 The search process of the OPC mining algorithm for a dataset with three attributes $\{a, b, c\}$

B. Recover True Clusters

In Section IV.B, we showed that an OPC corrupted by noise can be modelled by AOPC model. However, generating all AOPCs is NP-hard. Thus we propose an efficient greedy algorithm to generate significant AOPCs. Specifically, we propose a greedy algorithm to mine AOPCs through a recursive merging process. The time complexity of this algorithm is polynomial with respect to the number of OPCs found in Phase 1. Based on this basic algorithm, we propose an enhanced version with a hierarchical merge scheme which is much faster. Experiment demonstrates that the AOPCs generated by the enhanced algorithm are as significant as those found by the basic algorithm.

1) Mine AOPCs by Merging

The basic process of Phase 2 is to merge smaller AOPCs (initially OPCs) into bigger ones. To merge two AOPCs, we first take the union of their object sets then construct a common super-order of their orders as the consensus order of the new AOPC. Among all common super-orders, the one with the highest support is selected. The merge result is a valid AOPC only if both consistency and supporting criteria in

DEFINITION 4.1 are satisfied. To verify them, we need to (1) compute the LCS between the super-order and the order of every object in the union to check whether the length of the LCS is at least δ_c percent of the length of the super-order; (2) confirm that the super-order is supported by at least δ_s percent of the objects in the union. Since the merging process starts from valid OPCs found in Phase 1, the sizes of all AOPCs found are at least s_{min} and their order lengths are at least l_{min}. The computational complexity to fully test whether a pair of AOPCs can be merged is $O(s_{min} \times l_{min}^2)$. When the number of AOPCs is large, doing this for every AOPC pair would be very time consuming. Therefore, we propose a *prefiltering* technique which can quickly exclude most AOPC pairs that cannot be merged.

2) *Prefiltering*

In this section, we present a technique called *prefiltering,* which can exclude most AOPC pairs that cannot be merged with time complexity $O(s_{min})$ for each AOPC pair. It is much faster than performing the full test discussed above which has time complexity $O(s_{min} \times l_{min}^2)$ for each pair. The effectiveness of prefiltering will be demonstrated in the experiment section. In the following discussion, if C denotes an AOPC, then $o(C)$ denotes the order of C and $core(C)$ denotes the core of C.

LEMMA 5.1 *If C is an AOPC, then all objects follow o(C) are in core(C).*

Proof: Suppose that C is generated from C_n and C_n is generated from C_{n-1}, …, and C_1 is generated from C_0, through a series of merges, where $C_0,\ldots,C_n$ are interim AOPCs and C_0 is an OPC obtained in Phase 1. Because C is generated from C_n, $o(C)$ is a super-order of $o(C_n)$. So any object follows $o(C)$ must also follow $o(C_n)$. By the same token, they follow $o(C_{n-1})$, …, $o(C_0)$ as well. On the other hand, since C_0 is an OPC, according to the assumption we made in Section II: *all OPCs returned are maximal*, any object follows $o(C_0)$ is in C_0. Since no object is removed during any merge operation, any object in C_0 is in C. So any object follows $o(C)$ is in C, thus in $core(C)$. □

THEOREM 5.2 *A necessary condition under which two given AOPCs C_1 and C_2 can be merged is:*

$$\frac{|core(C_1) \cap core(C_2)|}{|C_1 \cup C_2|} > \delta_s \quad (3)$$

Proof: Suppose that C_1 and C_2 can be merged, and C is the AOPC after merge. For any object in $core(C)$, it follows $o(C)$. Since $o(C)$ is a super-order of both $o(C_1)$ and $o(C_2)$, this object follows $o(C_1)$ and $o(C_2)$ as well. From LEMMA 5.1, we know that this object is in $core(C_1)$ and $core(C_2)$ at the same time. Thus,

$$|core(C)| \le |core(C_1) \cap core(C_2)| \quad (4)$$

During the merge, the object set of C is generated by taking the union of the object sets of C_1 and C_2. Thus,

$$|C| = |C_1 \cup C_2| \quad (5)$$

Finally, according to the definition of AOPC, we have

$$\frac{core(C)}{|C|} > \delta_s \quad (6)$$

Therefore, Eq. (4) – Eq. (6) imply Eq. (3) holds. □

Based on THEOREM 5.2, we propose a linear time test called FILTER_TEST for each AOPC pair as shown in Figure 4. Each AOPC pair will first take this test before the merge routine. Those pairs failing this test will not be merged. Since this test only requires an intersection and a union operation of two sets, the time complexity is linear with respect to the size of the AOPCs. In addition to FILTER_TEST, COROLLARY 5.3 suggests that we should exclude an AOPC from future merge attempt if its FILTER_TEST with all other AOPCs fail. With FILTER_TEST, the algorithm can speedup substantially which we will show in the experiment section.

COROLLARY 5.3 *During the merge process, if the FILTER_TEST for an AOPC C with every other AOPC fails, C need not be considered for future merge.*

Proof: From THEOREM 5.2 we know that the core of the newly merged AOPC is a subset of the intersection of the cores of two AOPCs before merge. After each subsequent merge, the size of the core decreases while the size of the AOPC increases. Thus, the ratio at the left hand side of Eq. (3) decreases monotonically after each merge. Once it fails below δ_s when we do FILTER_TEST between C and any other AOPC, there is no need to do the FILTER_TEST for C in the future. This is because all new AOPCs in future are generated by merging current AOPCs. Thus it is impossible for C to be merged with another AOPC in the future. □

```
Input
• $\delta_s$, two AOPCs $C_1$ and $C_2$
Output
• succeed or fail
FILTER_TEST($C_1$, $C_2$)
1. if $size(core(C_1 \cap C_2))/size(C_1 \cup C_2) > \delta_s$, then
2.   return succeed;
3. else
4.   return fail;
5. end if;
```

Fig. 4 FILTER_TEST

3) *The Basic Algorithm*

We propose a greedy algorithm consisting of a series of iterations. The algorithm selects and merges the best pair of AOPCs at each iteration. If two AOPCs C_1 and C_2 pass both the FILTER_TEST and the full test, we compute their *adhesion* (AD value) as follows:

$$AD(C_1, C_2) = \frac{|core(C)|}{|C|} \times \frac{|LCS(o(C_1), o(C_2))|}{|o(C)|} \quad (7)$$

where C denotes the resulting AOPC, should C_1 and C_2 be merged. The intuition behind this is that the more the objects supporting the new super-order and the more similar $o(C_1)$ and $o(C_2)$ are, the more likely C_1 and C_2 should be merged.

Initially, the algorithm starts from the OPCs found in Phase 1. It prefilters most AOPC pairs that cannot be merged. For the rest pairs, it computes the adhesion for those pass the full test. Among them, the AOPC pair with the highest adhesion is selected to merge into a new AOPC. At each subsequent iteration, the above operations are only need to be done between the newly generated AOPC and other existing AOPCs. This process iterates until no new AOPC can be generated. The pseudo code is shown in Figure 5.

To analyse the complexity of this algorithm, let us assume that totally N OPCs are found in Phase 1. Before the first merge, the algorithm needs to do an operation on each AOPC pair, with totally $O(N^2)$ operations. Each operation can be either a prefiltering test or a full test (only if the pair passes the prefiltering test, the full test is needed). After that, at each iteration, the operations are only need to be done between the newly generated AOPC and other existing AOPCs. So there are totally $O(N)$ operations at each iteration. To find the largest AD value in an efficient way, we keep all AD values in a priority queue. The complexity of pushing and popping of this queue is $O(\lg N)$ at each iteration. Since each merge decreases the number of AOPCs by one, there are at most $O(N)$ iterations. So the complexity of this algorithm is $O(N^2m+N^2(\lg N+m))) = O(N^2(\lg N+m))$, where m denotes the *average* cost of a single operation for each AOPC pair. If a pair can be prefiltered, then this cost is only $O(s_{min})$, otherwise it is $O(s_{min}\times l_{min}^2)$, where s_{min} and l_{min} define the valid OPC in Phase 1. Since most pairs can be prefiltered (we will demonstrate this in the experiment section), m is close to $O(s_{min})$. So the average complexity of this algorithm is $O(s_{min}\times N^2\lg N)$ practically.

4) Hierarchical Merging

A possible way to speed up the algorithm is to use a hierarchical merge scheme. In such a scheme, we first partition the OPCs generated by Phase 1 into groups. There are 2^n groups at the nth level. Starting from Level n, we run the basic algorithm in each group. Then we take the union of the resulting AOPCs for each pair of sibling groups and proceed to Level n-1. The basic algorithm is to run repeatedly in each new group and generate AOPCs that will be used as the initial input to Level n-2. This procedure repeats until only one group is left as illustrated by Figure 6.

The time complexity of this scheme is $2p^{2n+2}/(2p^2-1)$ times the cost of the basic algorithm, where n is the number of hierarchical levels and p is an *average* fraction of AOPCs remaining after merge for each group, i.e. the ratio between the numbers of AOPCs after and before merge. The analysis details can be found in the appendix. This result suggests that a smaller p value makes the hierarchical merging scheme more effective. Therefore, it is desirable to group AOPCs that are likely to be merged so that the number of AOPCs to be carried to the next level is less. Thus when initially partitioning the OPCs at Level n, we want to put similar OPCs in the same group. As we discussed in Section V.A, the search method and the FIFO queue employed in Phase 1 naturally support this objective. The pseudo code of the hierarchical merge algorithm is shown in Figure 7.

```
Phase 2 (Basic)
Input
• $\delta_c$, $\delta_s$
• FIFO queue $Q$ contains OPCs found in Phase 1
Output
• A set of AOPCs
Algorithm
1.  $AD := \Phi$;
2.  for $(C_i, C_j) \in Q\times Q$, do
3.    if FILTER_TEST($C_i$, $C_j$)=succeed, then
4.      if FULL_TEST($C_i$, $C_j$)=succeed, then
5.        $AD$.push($AD(C_i, C_j)$);
6.      end if;
7.    end if;
8.  end if;
9.  while !$AD$.empty(), do
10.         merge ($C_i$, $C_j$) with the max $AD$ to $C$;
11.         $Q$.push($C$);
12.         $AD$.pop();
13.         $Q$.delete($C_i$); $Q$.delete($C_j$);
14.         for $C_i \in Q$, do
15.           if FILTER_TEST($C_i$, $C$)=succeed, then
16.             if FULL_TEST($C_i$, $C$)=succeed, then
17.               $AD$.push($AD(C_i, C)$);
18.             end if;
19.           end if;
20.         end for;
21.       end while;
Subroutine: FULL_TEST($C_i$, $C_j$)
1.  vote and determine super-order $o$;
2.  s := 0;
3.  for any $x$ in $C_i \cup C_j$, do
4.    $l := LCS(o(x), o)$.length;
5.    if $l \le o$.length$\times\delta_c$, then
6.      return fail;
7.    else if $l = o$.length, then
8.      $s := s+1$;
9.    end if;
10. end for;
11. if $s \le |C_i \cup C_j|\times\delta_s$, then
12.   return fail;
13. else
14.   return succeed;
end if;
```

Fig. 5 Basic algorithm of Phase 2

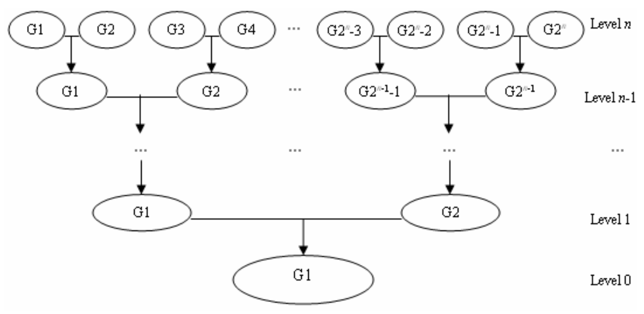

Fig. 6 n-level hierarchical merging scheme

Phase 2 (Hierarchical merge)
Input
- δ_c, δ_s, n (hierarchical levels)
- FIFO queue Q contains OPCs found in Phase 1

Output
- A set of AOPCs

Algorithm:

```
1.  for $i$ := 0 to $2^n$-1, do
2.    $G_i$ := $Q[i \times Q.\text{size}/2^n, (i+1) \times Q.\text{size}/2^n - 1]$;
3.  end for;
4.  for $l$ := $n$ downto 1, do  /* for each level */
5.    for any $i$, do
6.      run basic algorithm for $G_i$;
7.    end for;
8.    for $i$ := 0 to $2^{l-1}$, do
9.      merge $G_{2i}$ with $G_{2i+1}$;
10.     end for;
11.   end for;
```

Fig. 7 Enhanced algorithm of Phase 2

VI. Experiments

In this section, we study the performance of our algorithm through a series of experiments. To make the experiment results more realistic and thus convincing, we conduct all experiments on a real gene expression dataset. This dataset is the yeast cell cycle data from [9]. Each row of this dataset records the expression levels across 18 time points for a gene. Totally 799 genes are in this dataset. All experiments were run on a 3.4GHz Dell PC with 2G memory.

A. Choose Parameters Properly

First, we study the influence of the input parameters to the mining results and provide some guidance on how to choose them properly. There are totally four parameters that need to be specified in our algorithm. In Phase 1, s_{min} and l_{min} are set to define the valid OPC. In Phase 2, δ_c, δ_s are set to define the AOPC. In principle, the optimal values of these parameters depend on the size and shape of the salient clusters and the level and distribution of noise in the dataset. For our dataset, we use s_{min}=60, l_{min}=5 and found 682 OPCs in 495 seconds. As for δ_c and δ_s, they are often set to some moderate level to control the noise tolerance. As a rule of thumb, δ_s should be no larger than 0.5, otherwise it is too strict to prevent AOPCs from being merged; δ_c should be in [0.6, 0.8], since a too small value tends to include too much noise and thus decreases the significance of the results. Figure 8 shows the number of AOPCs found and the runtime under several (δ_c, δ_s) settings. Note that this result is without hierarchical merging. We can see that, as δ_c and δ_s decrease, more merges are performed (thus fewer AOPCs are returned) and the runtime is longer.

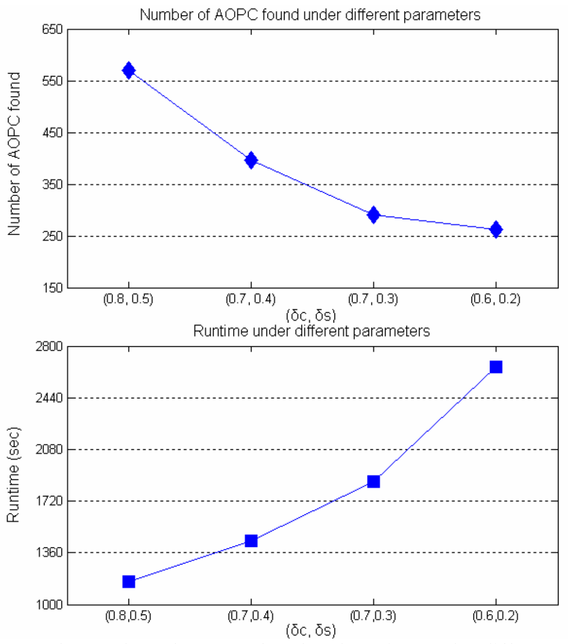

Fig. 8 The number of AOPCs found and runtime under different (δc, δs) settings, initially 682 OPCs

B. Efficiency of Optimizations

The experiments in this section are run with δ_c=0.6, δ_s=0.2. We ran Phase 2 with and without prefiltering, and with varying levels of hierarchical merging from 0 to 6, which correspond to 1, 2, 4, 8, 16, 32 and 64 leaf groups after initial partition. Note that the case of level 0 is essentially the basic algorithm. The result is shown in Figure 9, where #Group shows the number of initial groups. From the result we can see that the hierarchical merging with initially 64 groups can have an order of magnitude speedup over the basic algorithm. Plus, for all cases, prefiltering can also expedite the execution substantially. For the 64-group initial partition, we also compare the results of terminating the algorithm when 4, 2 and 1 group(s) are left respectively. The result is shown in Table II. The set of AOPCs when there are 4 groups left does not differ much from that of the final outcome (i.e., 1 group left). But the running time differs by half. This is because few new AOPCs are generated during the last two merging levels. (Most merge attempts fail to generate new AOPCs.) This fact suggests that we can consider terminating the algorithm earlier to gain more speed advantage.

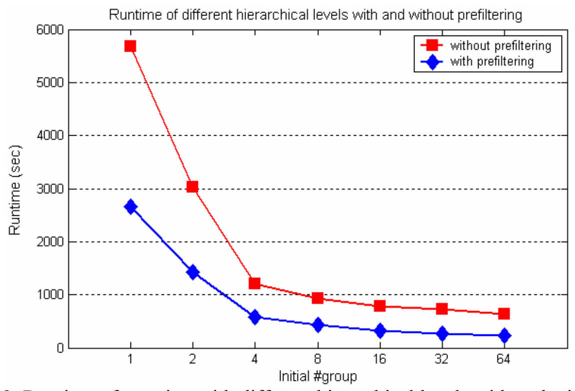

Fig. 9 Runtime of merging with different hierarchical levels with and without prefiltering

Figure 10 gives the percentage of AOPC pairs that are prefiltered under different (δ_c, δ_s) settings. For all cases, more than half of the AOPC pairs were prefiltered. The stricter the thresholds are, the more pairs that can be prefiltered. So the prefiltering stage plays an important role in speeding up the algorithm. The quality comparison of the results with and without hierarchical merging is discussed in next section.

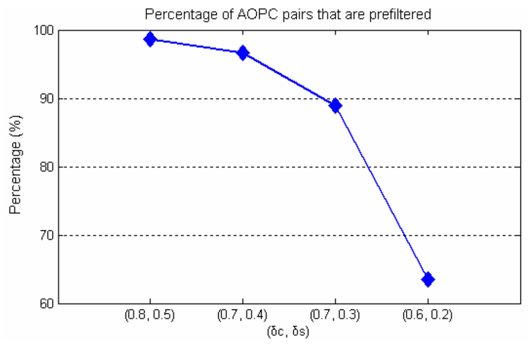

Fig. 10 Percentage of AOPC pairs prefiltered under different (δc, δs) settings

TABLE II
RESULTS WITH DIFFERENT STOP CRITERIA

#Groups when stop	Runtime(s)	#Cluster Found
4	96	285
2	142	276
1	229	267

C. *Significance of the Result*

Finally, we evaluate the quality of AOPCs found by our algorithm. We compare their significance with the OPCs found in Phase 1, since these are the result under the strict model. As mentioned in Section VI.A, with s_{min}=60, l_{min}=5, totally 682 OPCs were found. While with δ_c=0.6, δ_s=0.2, the basic algorithm found 262 AOPCs and the enhanced algorithm with 64-group initial partition found 267 AOPCs. For each cluster, we evaluate its biological significance by calculating its association strength with different gene categories according to the Fisher's exact test [10]. The known gene categories are based on gene ontology information from [11]. Smaller p-value indicates stronger association and thus is more significant. For each cluster, we record the number of categories it strongly associates with (p-value $\leq 10^{-9}$). Table III summarizes the distribution of this number for each method. 49 out of 262 AOPCs by the basic algorithm have more than six strongly associated categories; this number changes to 50 out of 267 by the hierarchical merging algorithm with 64-group initial partition. But under OPC model, there are only 14 out of 682 OPCs have such significance. The advantage of the AOPC model is even more prominent if a stronger association category cut-off is used. Moreover, there is little difference in the significance of the result of basic algorithm and that of hierarchical merging. This means that the hierarchical merging can speed up the mining process without scarifying the result significance. Note that the number of AOPCs is much smaller than that of OPCs, thus the percentage of high quality clusters of AOPC model is much higher than that of OPC model.

We also trace the merge process of an AOPC and compare its biological significance with all the six OPCs it is merged from. This result is summarized in Table IV. The AOPC has stronger associations to 5-7 categories (highlighted by bold fonts) than each OPC individually. This demonstrates that the AOPC found by our algorithm is biologically more significant than every single OPC it is merged from. Thus the merging process is an effective way to discover more significant clusters.

VII. CONCLUSIONS

In this paper, we study the problem of subspace OPC mining with noise tolerance. Due to its challenging nature, no previous work has been done on this topic. In this paper, we propose a new model called approximate order preserving cluster (AOPC). This model is proven to be more robust to noise than OPC model. In addition to its robustness, this model also has several good properties which allow us to mine the AOPCs using an efficient greedy algorithm. We propose a prefiltering technique which can quickly exclude most AOPC pairs that cannot be merged. We also propose a hierarchical merging scheme to further improve the execution time. Experiments on real gene expression data demonstrate that these techniques are efficient to speed up the mining process and the AOPCs are more biologically significant than the OPCs. Note that, although our experiments are run on gene expression data, the method presented in this paper can be used widely in many applications beyond gene expression analysis.

REFERENCES

[1] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: a review", ACM SIGKDD Explorations Newsletter, Volume 6, Issue 1, 2004, pp. 90-105.

[2] A. Ben-Dor, B. Chor, R.M. Karp and Z. Yakhini, "Discovering local structure in gene expression data: the order-preserving submatrix problem", Journal of Computational Biology 10(3/4), 2003, pp. 373-384.

[3] Y. Cheng and G.Church, "Biclustering of expression data", In Proceedings of the 8th International Conference on Intelligent System for Molecular Biology, 2000.

[4] H. Wang, W. Wang, J. Yang and P. Yu, "Clustering by pattern similarity in large data sets", In Proceedings of the *ACM SIGMOD*

International Conference on Management of Data (SIGMOD), pp. 394-405, 2002.

[5] X. Xu, Y. Liu, K. H. Tung and W. Wang, "Mining shifting-and-scaling co-regulation patterns on gene expression profiles", In *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE)*, pp. 89-98, 2006.

[6] J.Z. Liu and W. Wang, "OP-Cluster: Clustering by tendency in high dimensional space", In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 187-194.

[7] J.Z. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel and J. Prins, "Mining Approximate frequent itemset in the presence of noise: algorithm and analysis", In *Proceedings of the 6th SIAM Conference on Data Mining (SDM)*, pp. 405-416, 2006.

[8] L. Bergroth, H. Hakonen and T. Raita, "A survey of longest common subsequence algorithms", SPIRE, pp 39-48, 2000.

[9] Spellman et al., "Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization", Molecular Biology of the Cell 9, 1998, pp. 3273-3297.

[10] Fisher, R.A.(1922), "On the interpretation of $\chi 2$ from contingency tables, and the calculation of P", *Journal of the Royal Statistical Society* 85(1): pp. 87-94.

[11] Gene ontology consortium, www.geneontology.org.

APPENDIX

Suppose Phase 2 starts from N OPCs found in Phase 1 and the time used by the basic algorithm is T_b. As discussed in Section V.B.3, for fixed s_{min}, $T_b = O(N^2 \lg N)$, thus we can write T_b as $T_b = c \times N^2 \lg N$, where c is a constant related to s_{min}. In the following discussion, p denotes the ***average*** percentage of AOPCs remaining after merging in each group, i.e. the ratio between the numbers of AOPCs after and before merge. The total time used by the hierarchical version and the time used at Level n are denoted as T_h and T_n respectively. We now derive the ***average*** time used by the hierarchical version with n levels. We start from Level n.

At Level n, there are 2^n groups each contains $N/2^n$ OPCs. Each group consumes $c \times (N/2^n)^2 \lg(N/2^n))$ time for the basic algorithm. Thu we have:

$$\begin{aligned} T_n &= 2^n \times c \times (N/2^n)^2 \lg(N/2^n) \\ &= c \times N^2 (\lg N - \lg 2^n)/2^n \qquad \text{(a.1)} \\ &< c \times N^2 \lg N / 2^n = T_b / 2^n \end{aligned}$$

When the merge at Level n terminates, each group has $p \times N/2^n$ AOPCs. Then the algorithm proceeds to Level n-1, two sibling groups are merged to one. Now we have 2^{n-1} groups, each contains $p \times N/2^{n-1}$ AOPCs. Similarly, the time used in each group is smaller than $p^2 \times T/(2^{n-1})^2$, thus we have:

$$T_{n-1} < 2^{n-1} \times p^2 \times T/(2^{n-1})^2 = p^2 \times T_b/2^{n-1} \quad \text{(a.2)}$$

For Level n-k, we have:

$$T_{n-k} < p^{2k} \times T_b/2^{n-k}, \ k = 0, 1, \ldots, n \quad \text{(a.3)}$$

Taking the sum of T_k for all k = 0, 1, …, n, we get the total time used by the hierarchical merging scheme:

$$\begin{aligned} T_h &= \sum_{k=0}^{n} T_k < \sum_{k=0}^{n} p^{2k} \times \frac{T_b}{2^{n-k}} \\ &= \frac{T_b}{2^n} \times \sum_{k=0}^{n} (2p^2)^k \qquad \text{(a.4)} \\ &= \frac{T_b}{2^n} \times \frac{(2p^2)^{n+1} - 1}{2p^2 - 1} < T_b \times \frac{2p^{2n+2}}{2p^2 - 1} \end{aligned}$$

Since $0<p<1$, as n increases, the dividend of Eq. (a.4) decreases faster than the denominator. Eq. (a.4) suggests that more hierarchical levels and smaller p make the advantage of the hierarchical merging scheme more prominent.

TABLE III
THE DISTRIBUTIONS OF CLUSTERS FOUND BY THREE METHODS IN DIFFERENT INTERVALS OF NUMBER OF STRONGLY ASSOCIATED GENE CATEGORIES. (#) BEHIND EACH METHOD INDICATES THE TOTAL NUMBER OF CLUSTERS FOUND BY IT

#Gene Categories A Cluster Strongly Associates	**≤5**	**>5**	**>6**	**>7**	**>8**	**>9**	**>10**
OPC (682)	574	108	14	4	2	2	0
AOPC found by the basic algorithm (262)	161	101	49	29	17	9	6
AOPC found by hierarchical merge, initially 64 groups (267)	170	97	50	27	17	10	6

TABLE IV
P-VALUES FOR EACH GENE CATEGORY OF A AOPC FOUND BY OUR ALGORITHM AND THE SIX OPCS THE AOPCS WAS MERGED FROM; (#) BEHIND EACH CLUSTER INDICATES ITS SIZE

Gene Categories	**AOPC(134)**	**OPC_1(69)**	**OPC_2(71)**	**OPC_3(69)**	**OPC_4(73)**	**OPC_5(77)**	**OPC_6(65)**
cell	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	**10^{-14}**
bud	10^{-11}	**10^{-4}**	**10^{-7}**	**10^{-10}**	**10^{-4}**	**10^{-3}**	**10^{-5}**
intracellular	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}
cytoplasm	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}	10^{-15}
nucleus	10^{-15}	**10^{-8}**	**10^{-10}**	**10^{-11}**	**10^{-11}**	10^{-15}	**10^{-11}**
membrane	10^{-15}	10^{-15}	**10^{-13}**	**10^{-11}**	**10^{-15}**	**10^{-13}**	**10^{-12}**
integral	10^{-11}	**10^{-10}**	**10^{-6}**	**10^{-10}**	**10^{-8}**	**10^{-4}**	**10^{-7}**
plasma	10^{-13}	**10^{-6}**	**10^{-5}**	**10^{-6}**	**10^{-7}**	**10^{-7}**	**10^{-7}**
sensu Fungi	10^{-10}	**10^{-4}**	**10^{-6}**	**10^{-6}**	**10^{-2}**	**10^{-3}**	**10^{-7}**