Incremental Subspace Clustering over Multiple Data Streams

Qi Zhang Jinze Liu Wei Wang
University of North Carolina, Chapel Hill, NC 27599
{zhangq, liuj, weiwang}@cs.unc.edu

Abstract

Data streams are often locally correlated, with a subset of streams exhibiting coherent patterns over a subset of time points. Subspace clustering can discover clusters of objects in different subspaces. However, traditional subspace clustering algorithms for static data sets are not readily used for incremental clustering, and is very expensive for frequent re-clustering over dynamically changing stream data. In this paper, we present an efficient incremental subspace clustering algorithm for multiple streams over sliding windows. Our algorithm detects all the δ-CC-$Clusters$, which capture the coherent changing patterns among a set of streams over a set of time points. δ-CC-$Clusters$ are incrementally generated by traversing a directed acyclic graph pDAG. We propose efficient insertion and deletion operations to update the pDAG dynamically. In addition, effective pruning techniques are applied to reduce the search space. Experiments on real data sets demonstrate the performance of our algorithm.

1 Introduction

Stream mining problem has attracted a lot of research interest due to emerging stream applications such as network monitoring, sensor networks, financial data analysis, etc. Different from static data set, stream data is dynamically changing, potentially infinite, and possibly with fast speed, which poses new challenges to mining algorithm design.

To find interesting patterns for multiple data streams evolving over time, clustering stream into groups which exhibit coherent patterns has attracted significant interest. Streams are often inherently correlated. For example, the traffic volumes of connections in the same network, the stock prices of related businesses, the environmental readings from nearby sensors often exhibit coherent or correlated patterns. Traditional distance-based clustering methods group together streams similar to each other. The distance between streams are evaluated over the entire time dimension. However, for real world data, the stream correlations are typically local and only to a subset of time points for a subgroup of streams. Therefore, subspace clustering is more effective for discovering patterns over any subgroup of streams and subset of time points. Stream data is dynamic. Subspace clustering algorithms for static data-sets cannot readily be used for incremental computation and maintenance in stream setting. Re-clustering whenever stream data is updated can be quite expensive especially for large window size or for the entire stream.

In this paper, we propose an efficient incremental subspace clustering algorithm. Our subspace clustering model δ-CC-$Cluster$ considers the coherent changing pattern of a subgroup of streams over a subset of time points. Our model exerts two constraints for the subspace cluster. 1) Direction Constraint: the offsets of all the streams between any two time points inside the cluster are of the same direction (increasing or decreasing), and 2) Difference Constraint: the maximum difference between the offsets of any two streams over two time points inside the cluster is bounded by δ. Based on the clustering model, we propose an efficient incremental algorithm for computing δ-CC-$Cluster$. The algorithm organizes the current patterns or potential future patterns into a directed acyclic graph $pDAG$ according to the Anti-monotone Property. Whenever a new time point is considered, the patterns in $pDAG$ are accessed and extended to obtain new patterns in a certain order so that search space can be greatly reduced. Other pruning techniques are also proposed to further improve the algorithm performance.

2 Related Work

Subspace clustering has been extensively studied [3, 4, 2, 1] in recent years. It finds clusters that exist in multiple, possibly overlapping subspaces within high dimensional data sets. In this section, we mainly review related work in δ-Clusters. δ-Clusters is a unique kind of subspace clusters that use coherence as the similarity measure for clustering objects over a subset of dimensions. It was first proposed in [5] to identify clusters of genes which exhibit coherent expression levels over subset of conditions. In [10], pScore is proposed as the metric to evaluate the coherence within a

1550-4786/07 $25.00 © 2007 IEEE
DOI 10.1109/ICDM.2007.100

subspace δ-cluster described by similar shifting and scaling patterns. Liu and Wang [8] proposed OP-Cluster, which defines the coherent ordering of values of a cluster of objects over a subset of attributes.

There are many algorithms for clustering data streams. We will only review the work in subspace clustering for data streams. Kontaki et al. [7] presented an algorithm for incremental subspace clustering over a set of data streams. They proposed α-clusters, and provided efficient algorithms for computing the α-clusters incrementally using a set of pruning techniques. However, α-clusters only consider the subset of consecutive points over the time point dimension. In this paper, we consider a subspace clustering model which considers any subset of time points inside the sliding window. Narahashi and Suzuki [9] proposed an incremental subspace clustering algorithm to analyze web access log data for detecting the hostile accesses. Their algorithm uses a CF tree to compress data and effectively detect the subspace clusters.

3 Model

Consider a set of streams $S = \{s_1, \ldots, s_i, \ldots, s_m\}$ over a sequence of time points $\langle t_1, t_2, \ldots, t_n \rangle$, where t_n is the current time point. We are interested in the patterns existing in the latest w time points $T = \langle t_{n-w+1}, \ldots, t_n \rangle$. The stream data over this sliding window of size w can be represented as matrix $W = (d_{i,j})_{m \times w}$, where the i^{th} row corresponds to stream s_i, the j^{th} column corresponds to the j^{th} time point inside the sliding window which is t_{n-w+j}, also denoted as t_j^W.

Given the sliding window data matrix $W = (d_{i,j})_{m \times w}$ over the stream set S and time point set T, we first define the *Offset-direction Constraint* which describes the coherent offset direction exhibited by a pair of streams s_{i_1}, $s_{i_2} \in S$ over a pair of time points $t_{j_1}^W$, $t_{j_2}^W \in T$.

Definition 3.1 *Offset-direction Constraint (Dir-C). Given sliding window data matrix W over stream set S and time point set T, an Offset-direction Constraint is defined on a 2-by-2 sub-matrix* $\begin{vmatrix} d_{i_1,j_1} & d_{i_1,j_2} \\ d_{i_2,j_1} & d_{i_2,j_2} \end{vmatrix}$ *of W on any stream pair s_{i_1}, $s_{i_2} \in S$, and any time point pair $t_{j_1}^W$, $t_{j_2}^W \in T$ which requires:*

$$sign(d_{i_1,j_2} - d_{i_1,j_1}) = sign(d_{i_2,j_2} - d_{i_2,j_1}). \quad (1)$$

With Offset-direction Constraint, we define the *Offset-difference Constraint* as follows:

Definition 3.2 *Offset-difference Constraint (Dif-C). Given sliding window data matrix W over stream set S and time point set T, an Offset-difference Constraint is defined on a 2-by-2 sub-matrix* $\begin{vmatrix} d_{i_1,j_1} & d_{i_1,j_2} \\ d_{i_2,j_1} & d_{i_2,j_2} \end{vmatrix}$ *of W on any stream pair s_{i_1}, $s_{i_2} \in S$, and any time point pair $t_{j_1}^W$, $t_{j_2}^W \in T$ which requires:*

$$|\Delta_{j_1,j_2}^{i_1} - \Delta_{j_1,j_2}^{i_2}| \leq \delta \quad (2)$$

To allow minor noises, we define

$$\Delta_{j,j'}^{i} = \begin{cases} d_{i,j} - d_{i,j'} & \text{if } |\mathrm{d_{i,j}} - \mathrm{d_{i,j'}}| > \gamma \\ 0 & \text{otherwise} \end{cases}$$

where δ, γ are user-defined thresholds.

Based on $Dir\text{-}C$ and $Dif\text{-}C$, we define our clustering model as follows:

Definition 3.3 *δ-CC-$Cluster$. Given sliding window data matrix W over stream set S and time point set T. For any subset of streams S_p, and any subset of time points T_p, (S_p, T_p) is a δ-CC-$Cluster$ iff. any 2-by-2 sub-matrix in W_{S_p,T_p} satisfies $Dir\text{-}C$ and $Dif\text{-}C$.*

In fact, $Dif\text{-}C$ is the same requirement as in δ-$pCluster$ [10] and restricts p-$Score$ to be smaller than δ. δ-$pCluster$ controls the maximum difference between the offsets of any two streams for any pair of time points inside the cluster. However, it does not require the offsets to be in the same direction (increasing/decreasing). In fact, the offset direction becomes critical if we have a considerably large δ (in which case we want to relax δ to get more patterns). δ-CC-$Cluster$ controls both change in direction and maximum offset difference using $Dir\text{-}C$ and $Dif\text{-}C$, and therefore qualifies as both a δ-$pCluster$ and an OP-$Cluster$. As a result, the δ parameter can be safely and flexibly adjusted to get the patterns with desired quality of compactness.

In this paper, we consider δ-CC-$Cluster$ (S_p, T_p) as a *significant pattern* if it satisfies $|S_p| \geq nr$ and $|T_p| \geq nc$, where nr, nc are user defined thresholds for minimum number of streams (rows) and minimum number of time points (columns) required for a cluster.

4 Algorithms

Subspace clustering over data streams requires performing the clustering process in a sliding window over the streaming data. To avoid computationally intensive re-clustering, an incremental clustering algorithm is required.

4.1 pDAG and pTable

We maintain a directed acyclic graph $pDAG$ and a pattern table $pTable$ which record all the δ-CC-$Cluster$s for the current sliding window W as well as all the potential δ-CC-$Cluster$s ($\#rows \geq nr$ but $\#columns < nc$) for future sliding windows. Note that all clusters in the current window are not potential δ-CC-$Cluster$s (even if $\#rows \geq nr$). Trivially storing all the δ-CC-$Cluster$s

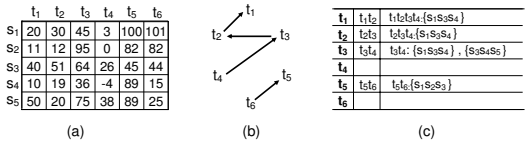

Figure 1. Initial sliding window and corresponding pDAG and pTable. (a) is the initial sliding window data matrix. (b) is the corresponding pDAG of (a). (c) is the corresponding pTable of (a). Here, $w = 6$, $\delta = 4$, $\gamma = 2$, $nr = 3$, $nc = 4$. The detailed initialization process is illustrated in Fig. 2.

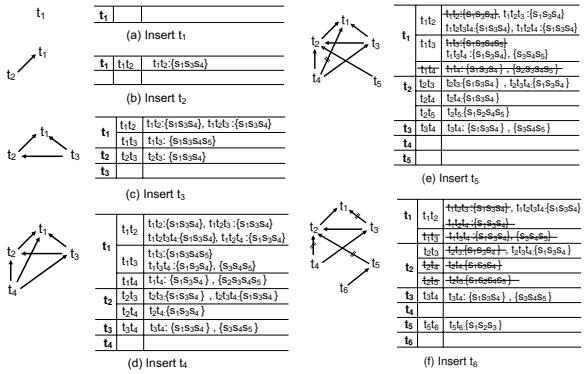

Figure 2. Initialization process for pDAG and pTable for the first sliding window. Here, $w = 6$, $\delta = 4$, $\gamma = 2$, $nr = 3$, $nc = 4$. The sliding window data matrix is in Fig.1(a). The final pDAG and pTable after initialization are in Fig.1(b), Fig.1(c).

is extremely inefficient for both memory usage and pattern growth. Therefore, carefully designed pruning techniques and pattern growth strategies are used to greatly reduce the search space and improve the performance.

In $pTable$, δ-CC-$Cluster$s and potential δ-CC-$Cluster$s (S_p, T_p) are organized according to the corresponding time sequence $T_p = \langle t_{i_1} \dots t_{i_k} \rangle$ $(t_{i_1} < \dots < t_{i_k})$ (We will use "pattern" to refer to any such (S_p, T_p) recorded inside $pTable$ in the following discussion). For example, $(S_{p_1}, T_p), \dots, (S_{p_i}, T_p)$ share entry $t_{i_1} \dots t_{i_k} : S_{p_1}, \dots, S_{p_i}$ in $pTable$ (see Fig.1(c), entry $t_3t_4 : \{s_1s_3s_4\}, \{s_3s_4s_5\}$). Each S_p satisfies $|S_p| \geq nr$. However, not every T_p has length $k \geq nc$. Note that even if the length of T_p is smaller than nc, the corresponding pattern could be potential δ-CC-$Cluster$s in the future. The pattern entry $t_{i_1} \dots t_{i_k} : S_{p_1}, \dots, S_{p_i}$ is indexed by the first time point t_{i_1} and first time pair $t_{i_1}t_{i_2}$ for easy manipulation in pattern growing process (see Fig.1(c)).

In $pDAG$, each node uniquely represents a time point t_i, $t_i \leq t_n$, where t_n is the current time point (Fig.1(b)). For any edge $t_j \rightarrow t_i$, we have $t_j > t_i$. Each path in $pDAG$ $t_{i_1} \leftarrow \dots \leftarrow t_{i_k}$ corresponds to an entry in $pTable$: $t_{i_1}t_{i_2}t_{i_k} : S_{p_1}, \dots, S_{p_i}$. For example in Fig.1(b), path $t_2 \leftarrow t_3 \leftarrow t_4$ corresponds to entry $t_2t_3t_4 : \{s_1s_3s_4\}$ in Fig.1(c). Note that not every pattern (S_p, T_p) with $|S_p| > nr$ is stored in $pTable$, therefore, not every pattern can find the corresponding path. For example, pattern $(s_1s_3s_4, \langle t_2t_4 \rangle)$ has 3 streams ($nc = 3$), but it is not in $pTable$ and does not have corresponding path in $pDAG$ since it is not a δ-CC-$Cluster$ and it is not possible to be one in the future. In addition, for time sequence T_1, T_2 corresponding to two entries in $pTable$, if $T_1 \subset T_2$, then the path in $pDAG$ for T_1 is a sub-path of the path for T_2. If we traverse $pDAG$ from terminal nodes which have no incoming edges, $pDAG$ guarantees we will meet all the patterns with growing time sequence patterns.

In the following section, we will describe in detail the algorithm of constructing and updating $pDAG$ and $pTable$ to incrementally generate the patterns. The algorithm has two phases. The initialization phase constructs $pDAG$ and $pTable$ for the initial sliding window W_0. After the sliding window is filled up, the incremental growing phase updates $pDAG$ and $pTable$ of the current sliding window W to obtain $pDAG$ and $pTable$ for the next sliding window W'. The brief routines of both phases are illustrated in Algo 1 and Algo 2.

Algorithm 1 Initialization($\hat{t_i}$)
Input $\hat{t_i}$: stream data column at current time point t_i

1: $Insert(\hat{t_i}, pDAG, pTable)$
2: $Prune(pDAG, pTable)$

Algorithm 2 IncConstruct($\hat{t_i}$)
Input $\hat{t_i}$: stream data column at current time point t_i

1: $Delete(\hat{t_{i-m}}, pDAG, pTable)$
2: $Insert(t_i, pDAG, pTable)$
3: $Prune(pDAG, pTable)$

4.2 Initialization

We begin with an example of how $pDAG$ and $pTable$ are constructed during the initialization phase (see Fig.1). Before proceeding, we first introduce *time-pair MDS* (maximal dimension set)[6]. A δ-CC-$Cluster$ $C_1 = (S_1, \langle t_i, t_j \rangle)$ is a time-pair MDS if there exists no δ-CC-$Cluster$ $C_2 = (S_2, \langle t_i, t_j \rangle)$ such that $S_1 \subset S_2$. Particularly, we denote $StreamSets(t_it_j)$ as the set $\{S_{i,j} | (S_{i,j}, \langle t_i, t_j \rangle)$ is a time-pair MDS$\}$. In general, for any $T_p \subseteq T$, we define $StreamSets(T_p)$ as $\{S_p | (S_p, T_p)$ is a δ-CC-$Cluster\}$. We also define

$$StreamSets(T_1) \widetilde{\bigcap} StreamSets(T_2) = \{S_k | \exists S_1, S_2 : (S_1 \in StreamSets(T_1)) \wedge (S_2 \in StreamSets(T_2)) \wedge (S_k = S_1 \bigcap S_2)\}.$$

In the following, we describe the process of building $pDAG$ and $pTable$ for the initial sliding window matrix W_0 (Fig.1(a)), with parameters $w = 6$, $\delta = 4$, $\gamma = 2$, $nr = 3$, $nc = 4$, $pDAG$ and $pTable$.

1. Initially, $pDAG$ and $pTable$ are empty.
2. When t_1 comes, node t_1 is inserted into $pDAG$. We then create an empty entry indexed by t_1 inside $pTable$ (Fig.2(a)).

3. When t_2 comes, we insert node t_2 into $pDAG$, and create an empty entry indexed by t_2 inside $pTable$ (Fig.2(b)). We compute the time-pair MDSs between t_2 with every other node in $pDAG$, *i.e.*, obtain $StreamSets(t_1t_2)$. We prune off any $S' \in StreamSets(t_1t_2)$ where $|S'| < nr$, then $StreamSets(t_1t_2) = \{s_1s_3s_4\}$. We add edge $t_2 \rightarrow t_1$ into $pDAG$, and add corresponding entry $t_1t_2 : \{s_1s_3s_4\}$ into $pTable$, which is indexed by t_1 and t_1t_2.
4. When t_3 comes, we insert node t_3 into $pDAG$, we create an empty entry indexed by t_3 inside $pTable$ (Fig.2(c)). We compute the time-pair MDS between t_3 with every other node in $pDAG$, *i.e.*, t_1 and t_2. The MDSs are not empty, thus we add edges $t_3 \rightarrow t_1$ and $t_3 \rightarrow t_2$ into $pDAG$, and insert $t_1t_3 : \{s_1s_3s_4s_5\}$, $t_2t_3 : \{s_1s_3s_4\}$ into $pTable$. Then we start growing the old patterns using t_3 by traversing through all the paths in $pDAG$ starting from t_3. For example, consider path $t_3 \rightarrow t_2 \rightarrow t_1$. Starting from t_3, we set $curStreamSets = \{S\}$, where S is the whole stream set, and then move to node t_2. At t_2, we get $StreamSets(t_2t_3)$ from $pTable$. We update $curStreamSets = curStreamSets \widetilde{\bigcap} StreamSets(t_2t_3)$, if $curStreamSets$ is not empty, then move to node t_1. At node t_1, we get $StreamSets(t_1t_3)$. Also, the path we have traversed is $t_2 \rightarrow t_1$. We get the corresponding $StreamsSets(t_1t_2)$ from $pTable$, and update $curStreamSets = curStreamSets \widetilde{\bigcap} StreamSets(t_1t_3) \widetilde{\bigcap} StreamsSets(t_1t_2)$. If $curStreamSets$ is not empty, we insert $t_1t_2t_3 : currentMDS$ into $pTable$, with index $t_1 : t_1t_2$. Since t_1 is already the end of the path, we stop. The traversal of the other path $t_3 \rightarrow t_1$ is similar.
5. When t_4 comes, we update $pTable$ and $pDAG$ in a similar way (Fig.2(d)).
6. When t_5 comes, we update $pTable$ and $pDAG$ in a similar way(Fig.2(e)). After the insertion is done, we prune out all the entries $T_p : StreamSets(T_p)$ indexed by t_1 where $|T_p| < 3$. Those patterns can at most be extended by 1 after t_6 comes. After pruning, there is no pattern which is indexed by t_1t_4. We also delete edge $t_4 \rightarrow t_1$ from $pDAG$.
7. When t_6 comes, similar insertion and pruning operations are performed (Fig.2(f)).

The Insert and Prune operations are formally defined in Algo 3 and Algo 4.

4.2.1 Insert Operation

Given sliding window W, the corresponding $pDAG$ and $pTable$, and the stream data $\hat{t_i}$ of the new-coming time point t_i, the Insertion operation grows the patterns in $pTable$ by traversing $pDAG$ to generate new patterns containing t_i, and update $pDAG$ and $pTable$. To scan through all the old patterns (S_j, T_j) in $pTable$ for pattern growth, we consider the following Anti-Monotone property:

Property 4.1 *Anti-Monotone Property: If there exists no pattern (S_j, T_j) which can grow with t_i, i.e., $StreamSets(T_jt_i) = \Phi$, then no pattern (S_k, T_k), $T_k \supset T_j$ can grow with t_i.*

Algorithm 3 Insert($\hat{t_i}, pDAG, pTable$)
Input $\hat{t_i}$: stream data column at current time point t_i

```
1:  Insert node t_i into pDAG.
2:  Create empty entry indexed with t_i in pTable.
3:  for every node t_j ∈ pDAG do
4:    Compute MDSs for⟨t_j, t_i⟩, get StreamSets(t_jt_i)
5:    Prune out all S_ps where S_p ∈ StreamsSets(t_jt_i) and |S_p| < nr
6:    if StreamSets(t_jt_i) is not empty then
7:      Add edge t_i → t_j into pDAG
8:      Insert entry t_jt_i : StreamSets(t_jt_i) into pTable indexed by t_j :
        t_jt_i
9:    end if
10: end for
11: Start from t_i, traverse all the nodes in pDAG which is reachable from t_i. Set
    t* = t_i, curStreamSets = {S}, curPath = ⟨⟩.
12: for every node t_j where there exists t_j ← t* in pDAG do
13:   curPath = ⟨t_j, curPath⟩
14:   Look up pTable for StreamSets(t_it_j)
15:   Look up pTable for StreamSets(curPath), if |curPath| = 1, set
      StreamSets(curPath) = {S}
16:   Compute curStreamSets = curStreamSets ⋂̃
      Streamsets(t_it_j) ⋂̃ StreamSets(curPath)
17:   if curStreamSets is not empty then
18:     Insert ⟨curPath, t_i⟩ : curStreamSets into pTable
19:     if If t_j does not have outgoing edges in pDAG then
20:       break
21:     else
22:       t* = t_j, go to 11
23:     end if
24:   end if
25: end for
```

Algorithm 4 Prune($t_i, pDAG, pTable$)
Input t_i: current time point

```
1:  for every index time point t_j ∈ pDAG do
2:    for every entry T_p : StreamSets(T_p) indexed by t_j do
3:      if |T_p| + (t_j − t_1) + (t_w − t_i) < nc then
4:        Delete T_p : StreamSets(T_p) from pTable
5:        if there is no entry indexed by the same secondary index then
6:          Delete the corresponding edge of the secondary index from pDAG.
7:        end if
8:      end if
9:    end for
10: end for
```

This Anti-Monotone property determines our search strategy during pattern growth. The Insertion operation scans all the patterns in $pTable$ by traversing through all the paths in $pDAG$, where we always visit patterns (S_p, T_p) with shorter T_p first. The Anti-Monotone property guarantees that the search space is reduced during Insertion and all the necessary patterns in $pTable$ are examined and all the possible patterns are generated. In addition to completeness, the correctness is ensured by the following Lemma.

Lemma 4.1 *Consider sliding window W over stream set S and time point set T, the new incoming time point t_i, if Insert operation inserts an entry $\langle T_p, t_i\rangle : SS$, then for any pattern $(S_p, \langle T_p, t_i\rangle)$, $S_p \in SS$, we have $(S_p, \langle T_p, t_i\rangle)$ is a $\delta-CC-Cluster$, and $S_p \geq nr$.*

Suppose that entry $\langle T_p, t_i\rangle : SS$ is generated at node t_j. According to Algo 3, $SS = curStreamSets$ is computed at line 16. If $|T_p| = 1$, $SS = S \widetilde{\bigcap} StreamSets(t_jt_i) \widetilde{\bigcap} S = StreamSets(t_jt_i)$. Therefore, for any $S_p \in$

SS, $(S_p, \langle T_p, t_i \rangle)$ is the MDS for (t_j, t_i), thus it is a $\delta-$CC$-Cluster$. According to the pruning at line 5, we also have $S_p \geq nr$. If $|T_p| > 1$, let $T_p = \langle T'_p, t_j \rangle$, then $SS = SS' \widetilde{\bigcap} StreamSets(t_j t_i) \widetilde{\bigcap} StreamSets(T'_p)$. Here, SS' is generated at the previous node, and entry $\langle T'_p, t_i \rangle : SS'$ has been inserted to $pTable$. We only need to prove that $\forall S_p \in SS, t', t'' \in T_p \bigcup \{t_i\}$, $(S_p, \langle t', t'' \rangle)$ is a δ-CC-$Cluster$. First, if $t', t'' \in T_p$, it holds. Second, if t', t'' are t_i and t_j, it holds. Finally, if one of t', t'' is t_i, the other one is in T'_p, it also holds.

4.2.2 Prune Operation

The Prune operation prunes out all the patterns that are not or unable to be significant patterns, which reduces the search space for future pattern growth.

Some pruning criteria are applied inside the $Insert$ operation.

Pruning Criterion 1 During pattern growth in Algo 3, any generated MDS $(S_p, \langle t_j, t_i \rangle)$ for the new time point t_i and a previous time point t_j with $|S_p| \leq nr$ is pruned out.

Pruning Criterion 2 During the traversal of the $pDAG$ along a path, if at a node t_j, the patterns at t_j cannot grow with the new time point t_i, then stop moving forward along that path. This pruning criterion is according to the Anti-Monotone Property.

In Prune operation, we perform pruning according to the following critera.

Pruning Criterion 3 For any pattern (S_p, T_p) in $pTable$ where T_p starts with t_j, suppose that the first time point of the current sliding window is t_1, the sliding window size is w, the current time point is t_i, then (S_p, T_p) is not a potentially significant pattern if $|T_p| + (t_j - t_1) + (t_w - t_i) < nc$ (see Algo 4). If the sliding window is full, then $t_w = t_i$.

The current size of the pattern (S_p, T_p) along the time dimension is T_p. If the pattern starts at t_j with t_1 as the starting time point of the sliding window, then the pattern (S_p, T_p) can at most be extended by $(t_j - t_1) + (t_w - t_i)$ along the time dimension, where t_i is the current time point. Therefore, for pattern (S_p, T_p) to be able to be a future significant pattern, it has to satisfy $|T_p| + (t_j - t_1) + (t_w - t_i) \geq nc$.

Pruning Criterion 4 For any edge $e = t_i \rightarrow t_j$ in $pDAG$, if there are no entries indexed by e in $pTable$, then delete e.

There is no entry indexed by e since $\langle t_j, t_i \rangle$ is not in any significant or potentially significant pattern. Therefore, e can be deleted to reduce the search space.

4.3 Incremental Construction

After sliding window W is filled up, $pDAG$ and $pTable$ are incrementally updated according to Algo 2. The Delete operation deletes all the related patterns containing the oldest time point and all its incident edges from $pDAG$, which will be obselete. Then the Insert and Prune operations are performed as Initialization phase.

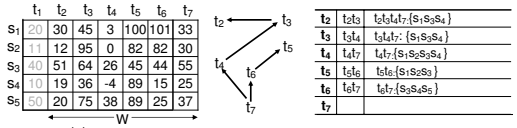

Figure 3. **Example of incremental construction: sliding window t_2-t_7 and corresponding $pDAG$ and $pTable$**

Fig.3 illustrates an example of incremental construction from sliding window over $\langle t_1, \ldots, t_6 \rangle$ (Fig.1(a)) to the next sliding window over $\langle t_2, \ldots, t_7 \rangle$ (Fig.3). The $pDAG$ and $pTable$ are after Delete, Insert and Prune operations.

5 Experiments

We performed our experiments on two real data sets 1) STOCK Data Set: 30 stocks are chosen from historical data of S&P 500 stocks[1], each has 253 values which represent the daily open prices of that stock throughout the year from May, 2006 till May, 2007. 2) Climate Data Set[2]: temperature data for 25 cities, each contains the daily temperature of the corresponding city throughout a year.

The performance of our algorithm is mainly dependent on the pruning efficiency obtained in a sliding window. Higher pruning efficiency results in faster incremental computations as the sliding window advances in time. The pruning efficiency is dependent on several factors. In order to measure the performance of our experiments, we used a sliding window of size 15 for STOCK data set (3 weeks) and 14 for climate data set (2 weeks). We used default parameters of $\delta = 4.0$, $\gamma = 0.2$, $nr = 5$ and $nc = 7$ for the stock data set, and $\delta = 0.3$, $\gamma = 0.002$, $nr = 4$, $nc = 7$ for the climate data set. In our experiments, we tested the performance of our algorithm by using default parameters for the datasets and varying one parameter at a time.

Minimum number of time point(*nc*): We expect higher pruning efficiency as nc increases. Fig. 4(a) and 4(f) compare the performance of our algorithm against the algorithm without pruning as nc varies. We observe that for larger nc, our algorithm achieves higher speedup as expected. Our algorithm also performs better in capturing larger patterns. Overall, for both data sets, our algorithm exhibits 2-18x speedup over the algorithm without pruning.

Stream count threshold nr: As nr increases, the number of patterns common to at least nr streams decreases, therefore increasing the pruning efficiency. Figs. 4(b), and 4(g) show the performance of our algorithm by varying nr. We

[1] http://biz.swcp.com/stocks/

[2] http://hprcc.unl.edu/data/index.php

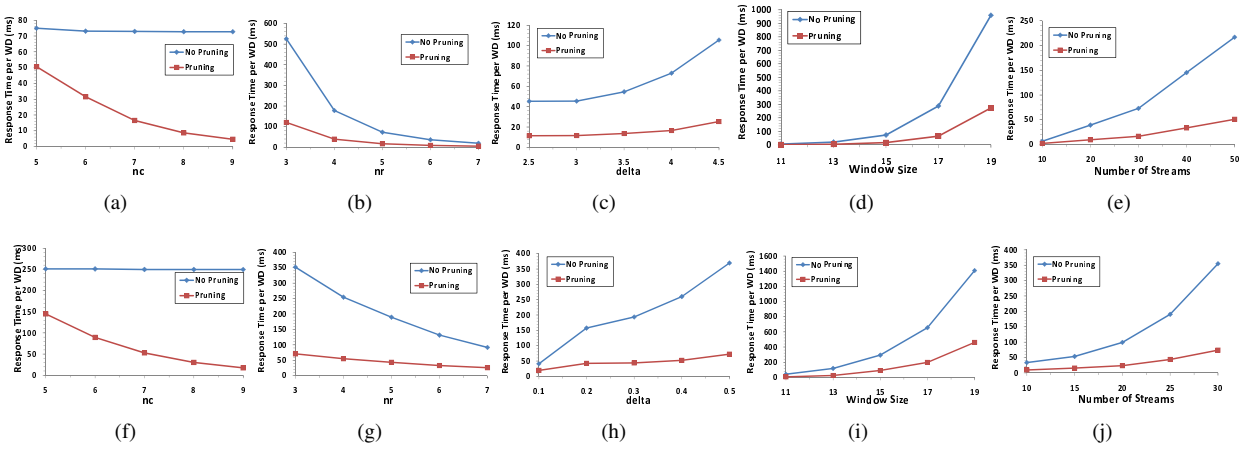

Figure 4. Performance on Stock (the first row) and Climate data (the second row)

observe that our algorithm is up to 4x faster on the stock data and 5x faster on climate data.

δ **threshold**: The δ threshold determines the maximum difference between two points in a pattern. As δ increases, we expect to identify more patterns satisfying the threshold. Therefore, the pruning efficiency decreases as δ increases and the overall performance decreases. Fig. 4(c), and Fig. 4(h) highlight the performance of our algorithm with varying δ, with a speedup of 4x in Fig. 4(c) and a speedup of 5x faster in Fig. 4(h).

Sliding window size: As the sliding window size increases, we expect higher number of patterns. Therefore, we expect the overall performance to decrease as the sliding window size increases and vice-versa. Fig. 4(d), and Fig. 4(i) indicate that our algorithm achieves 5x and 6x higher performance than without pruning. Besides, the response time for our algorithm increases slowly and scales better than the algorithm without pruning.

Number of streams: Similar to the sliding window size parameter, as the number of streams increases we expect higher number of patterns. Therefore the response time is expected to decrease as the number of streams increase. We tested the algorithms on Stock data sets of sizes 10, 20, 30, 40, 50 (Fig. 4(e)), and climate data sets of sizes 10, 15, 20, 25, 30 (Fig. 4(j)). Our algorithm with pruning techniques is up to 4x faster in Fig. 4(e) and up to 5x faster in Fig. 4(j).

6 Conclusion

We proposed an efficient incremental algorithm for subspace clustering of multiple streams over sliding windows. Our subspace clustering model δ-CC-$Cluster$ captures the coherent changing pattern of a subgroup of streams over a subset of time points. We maintain all the significant patterns and potential future significant patterns of the current sliding window and organize them into an acyclic graph $pDAG$ and incrementally update the $pDAG$ efficiently. The experiments over two real data sets demonstrate the efficiency and effectiveness of our algorithm.

References

[1] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park. Fast algorithms for projected clustering. *ACM SIGMOD International Conference on Management of Data*, pages 61–72, 1999.

[2] C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional spaces. *ACM SIGMOD International Conference on Management of Data*, pages 70–81, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining application. *The ACM International Conference on Management of Data*, pages 94–105, 1998.

[4] C.-H. Cheng, A. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. *The Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 84–93, 1999.

[5] Y. Cheng and G. Church. Biclustering of expression datas. *International Conference on Intelligent Systems for Molecular Biology*, pages 93–103, 2000.

[6] S. Goil, H. Nagesh, and A. Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets. *Technical Report CPDC-TR-9906-010, Northwestern University*, June 1999.

[7] M. Kontaki, A. Papadopoulos, and Y. Manolopouulos. Efficient incremental subspace clustering in data streams. *10th International Database Engineering and Applications Symposium (IDEAS'06)*, 2006.

[8] J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. *Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.

[9] M. Narahashi and E. Suzuki. Detecting hostile accesses through incremental subspace clustering. *IEEE/WIC International Conference on Web Intelligence (WI'03)*, 2003.

[10] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets. *ACM International Conference on Management of Data*, pages 394–405, 2002.