A Fast Algorithm for Subspace Clustering by Pattern Similarity

Haixun Wang   Fang Chu[1]   Wei Fan   Philip S. Yu   Jian Pei[2]

IBM T. J. Watson Research Center, {haixun,weifan,psyu}@us.ibm.com

[1]Dept. of Computer Science, Univ. of California, Los Angeles, fchu@cs.ucla.edu

[2]Dept. of Computer Science, SUNY Buffalo, jianpei@cse.buffalo.edu

Abstract

Unlike traditional clustering methods that focus on grouping objects with *similar values* on a set of dimensions, clustering by pattern similarity finds objects that exhibit a *coherent pattern* of rise and fall in subspaces. Pattern-based clustering extends the concept of traditional clustering and benefits a wide range of applications, including large scale scientific data analysis, target marketing, web usage analysis, etc. However, state-of-the-art pattern-based clustering methods (e.g., the pCluster algorithm) can only handle datasets of thousands of records, which makes them inappropriate for many real-life applications. Furthermore, besides the huge data volume, many data sets are also characterized by their sequentiality, for instance, customer purchase records and network event logs are usually modeled as data sequences. Hence, it becomes important to enable pattern-based clustering methods i) to handle large datasets, and ii) to discover pattern similarity embedded in data sequences. In this paper, we present a novel algorithm that offers this capability. Experimental results from both real life and synthetic datasets prove its effectiveness and efficiency.

1 Introduction

Clustering large datasets is a challenging data mining task with many real life applications. Much research has been devoted to the problem of finding subspace clusters [2, 3, 4, 7, 12]. Along this direction, we further extended the concept of clustering to focus on *pattern-based* similarity [21]. Several research work have since studied clustering based on pattern similarity [22, 15], as opposed to traditional *value-based* similarity.

These efforts represent a step forward in bringing the techniques closer to the demands of real life applications, but at the same time, they also introduced new challenges. For instance, the clustering models in use [21, 22, 15] are often too rigid to find objects that exhibit meaningful similarity, and also, the lack of an ef-

ficient algorithm makes the model impractical for large scale data. In this paper, we introduce a novel clustering model which is intuitive, capable of capturing subspace pattern similarity effectively, and is inducive to an efficient implementation.

1.1 Subspace Pattern Similarity

We present the concept of *subspace pattern similarity* by an example in Figure 1. We have three objects. Here, the X axis represents a set of conditions, and the Y axis represents object values under those conditions. In Figure 1(a), the similarity among the three objects are not visibly clear, until we study them under two subsets of conditions. In Figure 1(b), we find the same three objects form a shifting pattern in subspace $\{b, c, h, j, e\}$, and in Figure 1(c), a scaling pattern in subspace $\{f, d, a, g, i\}$.

This means, we should consider objects similar to each other as long as they manifest a coherent pattern in a certain subspace, regardless of whether their coordinate values in such subspaces are close or not. It also means many traditional distance functions, such as Euclidean, cannot effectively discover such similarity.

1.2 Applications

We motivate our work with applications in two important areas.

Analysis of Large Scientific Datasets. Scientific data sets often consist of many numerical columns. One such example is the gene expression data. DNA micro-arrays are an important breakthrough in experimental molecular biology, for they provide a powerful tool in exploring gene expression on a genome-wide scale. By quantifying the relative abundance of thousands of mRNA transcripts simultaneously, researchers can discover new functional relationships among a group of genes [6, 9].

Investigations show that more often than not, several genes contribute to one disease, which motivates researchers to identify genes whose expression levels rise

(a) Raw data: 3 objects, 10 columns

(b) A *Shifting Pattern* in subspace $\{b, c, h, j, e\}$

(c) A *Scaling Pattern* in subspace $\{f, d, a, g, i\}$
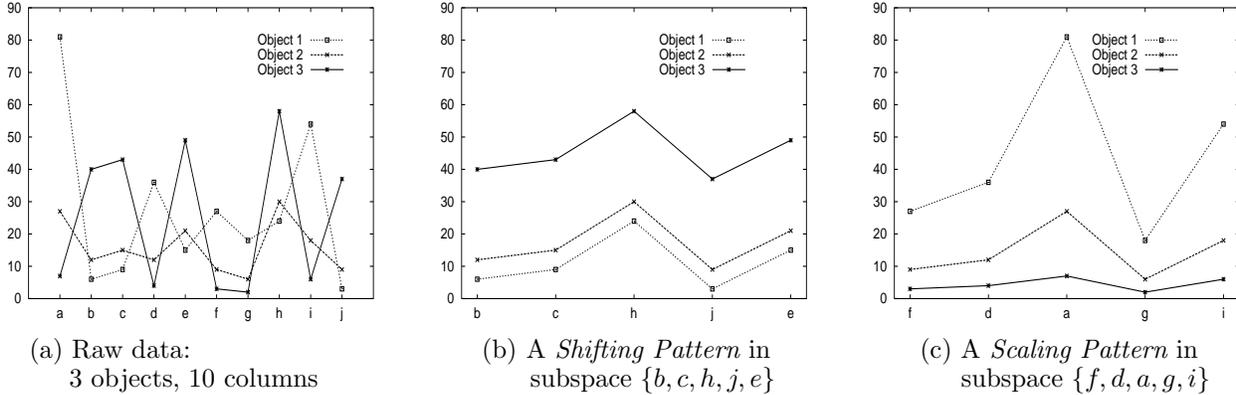
Figure 1: Objects form patterns in subspaces.

and fall coherently under a subset of conditions, that is, they exhibit fluctuation of a similar shape when conditions change [6, 9]. Table 1 shows that three genes, VPS8, CYS3, and EFB1, respond to certain environmental changes coherently.

More generally, with the DNA micro-array as an example, we argue that the following queries are of interest in scientific data analysis.

Example 1. *Counting*
How many genes whose expression level in sample CH1I is about 100 ± 5 units higher than that in CH2B, 280 ± 5 units higher than that in CH1D, and 75 ± 5 units higher than that in CH2I?

Example 2. *Clustering*
Find clusters of genes that exhibit coherent subspace patterns, given the following constraints: i) the subspace pattern has dimensionality higher than `minCols`*; and ii) the number of objects in the cluster is larger than* `minRows`*.*

Answering the above queries efficiently is important in discovering gene correlations [6, 9] from large scale DNA micro-array data. The *counting* problem of Example 1 seems easy to implement, yet it constitutes the most primitive operation in solving the *clustering* problem of Example 2, which is the focus of this paper.

Current database techniques cannot solve the above problems efficiently. Algorithms such as the pCluster [21] have been proposed to find clusters of objects that manifest coherent patterns. Unfortunately, they can only handle datasets containing no more than thousands of records.

DISCOVERY OF SEQUENTIAL PATTERNS. We use network event logs to demonstrate the need to find clusters based on sequential patterns in large datasets. A network system generates various events. We log each event, as well as the environment in which it occurs, into a database. Finding patterns in a large dataset of event

	CH1I	CH1B	CH1D	CH2I	CH2B	$\cdots$
VPS8	**401**	281	**120**	275	**298**
SSA1	401	292	109	580	238
SP07	228	290	48	285	224
EFB1	**318**	280	**37**	277	**215**
MDM10	538	272	266	277	236
CYS3	**322**	288	**41**	278	**219**
DEP1	317	272	40	273	232
NTG1	329	296	33	274	228
$\vdots$

Table 1: Expression data of Yeast genes

logs is important to the understanding of the temporal causal relationships among the events, which often provide actionable insights for determining problems in system management.

We focus on two attributes, Event and Timestamp (Table 2), of the log database. A network event pattern contains multiple events. For instance, a candidate pattern might be the following:

Example 3. *Sequential Pattern*
Event CiscoDCDLinkUp is followed by MLMStatusUp that is followed, in turn, by CiscoDCDLinkUp, under the constraint that the interval between the first two events is about 20 ± 2 seconds, and the interval between the 1st and 3rd events is about 40 ± 2 seconds.

Previous works [20, 19] have studied the problem of efficiently locating a given sequential pattern, however, finding all interesting sequential patterns is a difficult problem. A network event pattern becomes interesting if: i) it occurs frequently, and ii) it is non-trivial, meaning it contains a certain amount of events. The challenge here is to find such patterns efficiently.

Although seemingly different to the problem shown in Figure 1, finding patterns exhibited over the time in sequential data is closely related to finding coherent patterns in tabular data. It is another form of clustering by subspace pattern similarity: if we think of different

2

Event	Timestamp
⋮	⋮
CiscoDCDLinkUp	19:08:01
MLMSocketClose	19:08:07
MLMStatusUp	19:08:21
⋮	⋮
MiddleLayerManagerUp	19:08:37
CiscoDCDLinkUp	19:08:39
⋮	⋮

Table 2: A Stream of Events

type of events as conditions on the X axis of Figure 1, and their timestamp as the Y axis, then, we are actually looking for clusters of subsequences that exhibit (time) shifting patterns as in Figure 1(b).

1.3 Our Contributions.

This paper presents a novel approach to clustering datasets based on pattern similarity.

- We present a novel model for subspace pattern similarity. In comparison with previous models, the new model is intuitive for capturing subspace pattern similarity, and reduces computation complexity dramatically.

- We unify pattern similarity analysis in tabular data and pattern similarity analysis in sequential data into a single problem. Indeed, tabular data are transformed into their sequential form which is inducive to an efficient implementation.

- We present a scalable sequence-based method, Seq-Clus, for clustering by subspace pattern similarity. The technique outperforms all known state-of-the-art pattern clustering algorithms and makes it feasible to perform pattern similarity analysis on large dataset.

The rest of the paper is organized as follows. We introduce a novel distance function for measuring subspace pattern similarity in Section 2. Section 3 presents an efficient clustering algorithm based on a novel counting tree structure. Experiments and results are reported in Section 4. In Section 5, we review related work. We conclude in Section 6.

2 The Distance Function

The choice of distance functions has great implications on the meaning of similarity, and this is particularly important in subspace clustering because of computational complexity. Hence, we need a distance function that makes measuring of the similarity between two objects in high dimensional space meaningful and intuitive, and at the same time yields to an efficient implementation.

2.1 Tabular and Sequential Data

Finding objects that exhibit coherent patterns of rise and fall in a tabular dataset (e.g. Table 1) is similar to finding subsequences in a sequential dataset (e.g. Table 2). This indicates that we should unify the data representation of tabular and sequential datasets so that a single similarity model and algorithm can apply to both tabular and sequential datasets for clustering based on pattern similarity.

We use sequences to represent objects in a tabular dataset $\mathcal{D}$. We assume there is a total order among its attributes. For instance, let $\mathcal{A} = \{c_1, \cdots, c_n\}$ be the set of attributes. We assume $c_1 \prec \cdots \prec c_n$ is the total order. Thus, we can represent any object x by a sequence[1]:

$$\langle (c_1, x_{c_1}), \cdots, (c_n, x_{c_n}) \rangle$$

where x_{c_i} is the value of x in column c_i. We can then concatenate objects in $\mathcal{D}$ into a long sequence, which is a sequential representation of the tabular data.

After the conversion, pattern discovery on tabular datasets is no different from pattern discovery in a sequential dataset. For instance, in the Yeast DNA microarray, we can use the following sequence to represent a pattern:

$$\langle (\texttt{CH1D}, 0), (\texttt{CH2B}, 180), (\texttt{CH2I}, 205), (\texttt{CH1I}, 280) \rangle$$

In words, for genes that exhibit this pattern, their expression levels under condition CH2B, CH2I, and CH1I must be 180, 205, 280 units higher than that under CH1D.

2.2 Sequence-based Pattern Similarity

In this section, we propose a new distance measure that is capable of capturing subspace pattern similarity and is inducible to an efficient implementation.

Here we consider the shifting pattern of Figure 1(b) only, as scaling patterns are equivalent to shifting patterns after a logarithmic transformation of the data.

To tell whether two objects exhibit a shifting pattern in a given subspace $\mathcal{S}$, the simplest way is to *normalize* the two objects by subtracting $\bar{x}_s$ from each of their coordinate value x_i ($i \in \mathcal{S}$), where $\bar{x}_s$ is the average coordinate value of x in subspace $\mathcal{S}$. This, however, requires us to compute and keep track of $\bar{x}_s$ for each subspace $\mathcal{S}$. As there are as many as $2^{|\mathcal{A}|} - 1$ different ways of normalization, it makes the computation of such similarity model impractical for large datasets.

To find a distance function that yields to an efficient implementation, we choose an *arbitrary* dimension $k \in \mathcal{S}$ for normalization. We show that the choice of k has very limited impact on the similarity measure.

---

[1] We also use $\langle x_{c_1}, \cdots, x_{c_n} \rangle$ to represent x if no confusion arises.

More formally, given two objects x and y, a subspace $\mathcal{S}$, a dimension $k \in \mathcal{S}$, we define the *sequence-based distance* between x and y as follows:

$$dist_{k,\mathcal{S}}(x,y) = \max_{i \in \mathcal{S}} |(x_i - y_i) - (x_k - y_k)| \qquad (1)$$

Figure 2 demonstrates the intuition behind Eq (1). Let $\mathcal{S} = \{k, a, b, c\}$. With respect to dimension k, the distance between x and y in $\mathcal{S}$ is less than δ if the difference between x and y on any dimension of $\mathcal{S}$ is within $\Delta \pm \delta$, where Δ is the difference of x and y on dimension k.
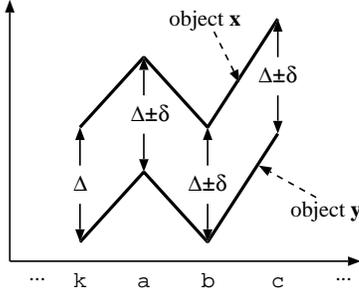
Figure 2: The meaning of $dist_{k,\mathcal{S}}(x,y) \leq \delta$.

Clearly, with a different choice of dimension k, we may find the distance between two objects different. However, such difference is bounded by a factor of 2.

Property 1. *For any two objects x, y, and a subspace $\mathcal{S}$, if $\exists k \in \mathcal{S}$ such that $dist_{k,\mathcal{S}}(x,y) \leq \delta$, then $\forall j \in \mathcal{S}$, $dist_{j,\mathcal{S}}(x,y) \leq 2\delta$.*

Proof.

$$
\begin{aligned}
dist_{j,\mathcal{S}}(x,y) &= \max_{i \in \mathcal{S}} |(x_i - y_i) - (x_j - y_j)| \\
&\leq \max_{i \in \mathcal{S}} |(x_i - y_i) - (x_k - y_k)| + \\
&\quad \max_{j \in \mathcal{S}} |(x_j - y_j) - (x_k - y_k)| \\
&\leq 2\delta
\end{aligned}
$$

$\square$

Since δ is but a user-defined threshold, Property 1 shows that Eq (1)'s capability of capturing pattern similarity does not depend on the choice of k, which can be an arbitrary dimension in $\mathcal{S}$. As a matter of fact, as long as we use a fixed dimension k for any given subspace $\mathcal{S}$, then, with a relaxed δ, we can always find those clusters discovered by Eq (1) where a different dimension k is used. This gives us great flexibility in defining and mining clusters based on subspace pattern similarity.

Problem Statement Our task is to find subspace clusters of objects where the distance between two objects is measured by Eq (1). Since in Eq (1), any dimension k is equally good in capturing subspace pattern similarity, we shall choose the one that leads to the most efficient computation.

3 The Clustering Algorithm

We define the concept of pattern and then we divide the pattern space into grids (Section 3.1). We then construct a tree structure which provides a compact summary of all of the frequent patterns in a data set (Section 3.2). We show that the tree structure enables us to find efficiently the number of occurrences of any specified pattern, or equivalently, the density of any cell in the grid (Section 3.3). A density and grid based clustering algorithm can then be applied to merge dense cells into clusters. Finally, we introduce an Apriori-like method to find clusters in any subspace (Section 3.4).

3.1 Pattern and Pattern Grids

Let $\mathcal{D}$ be a dataset in a multidimensional space $\mathcal{A}$. A pattern p is a tuple (T, δ), where δ is a distance threshold and T is an ordered sequence of (column, value) pairs, that is,

$$T = \langle (t_1, 0), (t_2, v_2), \cdots, (t_k, v_k) \rangle$$

where $t_i \in \mathcal{A}$, and $t_1 \prec \cdots \prec t_k$. Let $\mathcal{S} = \{t_1, \cdots, t_k\}$. An object $x \in \mathcal{D}$ exhibits pattern p in subspace $\mathcal{S}$ if

$$v_i - \delta \leq x_{t_i} - x_{t_1} \leq v_i + \delta, \qquad 1 \leq i \leq k. \qquad (2)$$

Apparently, if two objects $x, y \in \mathcal{D}$ are both instances of pattern $p = (T, \delta)$, then we have

$$dist_{k,\mathcal{S}}(x,y) \leq 2\delta.$$

In order to find clusters, we start with *high density patterns*: a pattern $p = (T, \delta)$ is of high density if given p, the number of objects that satisfy Eq (2) reaches a user-defined density threshold.
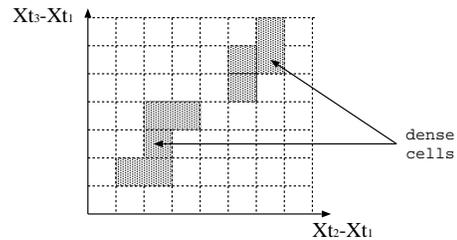
Figure 3: Pattern grids for subspace $\{t_1, t_2, t_3\}$

We discretize the dataset so that patterns fall into grids. For any given subspace $\mathcal{S}$, after we find the dense cells in $\mathcal{S}$, we use a grid and density based clustering algorithm to find the clusters (Figure 3).

The difficult part, however, lies in finding the the dense cells efficiently for all subspaces. The rest of this section deals with this issue.

4

3.2 The Counting Tree

The counting tree provides a compact summary of the dense patterns in a dataset. It is motivated by the suffix trie, which, given a string, indexes all of its substrings. Here, each record in the dataset is represented by a sequence, but sequences are different from strings, as we are interested in *non-contiguous* sub-sequence match, while suffix tries only handle *contiguous* substrings.

	c_1	c_2	c_3	c_4
x	4	3	0	2
y	3	4	1	3
z	1	2	3	1

Table 3: A dataset of 3 objects

Before we introduce the structure of the counting tree, we use an example to illustrate our purpose. Table 3 shows a dataset of 3 objects in a 4 dimensional space. We start with the *relevant subsequences* of each object.

Definition 1. *Relevant subsequences.*
The relevant subsequences of an object o in an n-dimensional space are:

$$x^i = \langle x_{i+1} - x_i, \cdots, x_n - x_i \rangle \quad 1 \le i < n$$

In relevant subsequence x^i, column c_i is used as the base for comparison. Assuming C is a cluster in subspace $\mathcal{S}$, wherein i is the minimal dimension, we shall search for C in dataset $\{x^i | \forall x \in \mathcal{D}\}$. In any such subspace $\mathcal{S}$, we use c_i as the base for comparison, in other words, c_i serves as the dimension k in Eq (1). As an example, the relevant subsequences of object z in Table 3 are:

	c_1	c_2	c_3	c_4
z^1		1	2	0
z^2			1	-1
z^3				-2

To create a counting tree for a dataset $\mathcal{D}$, for each object $z \in \mathcal{D}$, we insert its relevant subsequences into a tree structure. Also, assuming the insertion of a sequence, say z^1, ends up at node t in the tree (Figure 4), we increase the count associated with node t by 1.

More often than not, we are interested in patterns equal to or longer than a given size, say $\xi \ge 1$. A relevant subsequence whose length is shorter than ξ cannot contain patterns longer than ξ. Thus, if ξ is known beforehand, we only need to insert x^i where $1 \le i < n-\xi+1$ for each object x. Figure 4 shows the counting tree for the dataset of Table 3 where $\xi = 2$.

In the second step, we label each tree node t with a triple: $(ID^\vdash, ID^\dashv, Count)$. The first element of the triple, $ID^\vdash$, uniquely identifies node t, and the second element, $ID^\dashv$, is the largest $ID^\vdash$ of t's descendent nodes. The IDs are assigned by a depth-first traversal of the tree structure, during which we assign sequential numbers (starting form 0, which is assigned to the root node) to
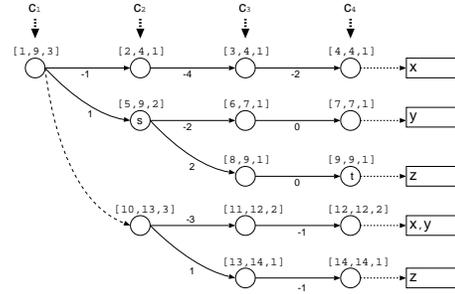
Figure 4: The Counting Tree

the nodes as they are encountered one by one. If t is a leaf node, then the 3rd element of the triple, $Count$, is the number of objects in t's object set, otherwise, it is the sum of the counts of its child nodes. Apparently, we can label a tree with a single depth-first traversal. Figure 4 shows a labeled tree for the sample dataset.

To count pattern occurrences using the tree structure, we introduce *counting lists*. For each column pair (c_i, c_j), $i < j$, and each possible value $v = x_j - x_i$ (after data discretization), we create a counting list (c_i, c_j, v). The counting lists are also constructed during the depth-first traversal. Suppose during the traversal, we encounter node t, which represents sequence element $x_j - x_i = v$. Assuming t is to be labeled $(ID^\vdash, ID^\dashv, cnt)$, and the last element of counting list (c_i, c_j, v) is $(\_, \_, cnt')$, we append a new element $(ID^\vdash, ID^\dashv, cnt + cnt')$ into the list[2].

link head		list of node labels
$\cdots$		$\cdots$
$(c_1, c_3, -4)$	$\Rightarrow$	$[3, 4, 1]$
$\cdots$		$\cdots$
$(c_1, c_4, -2)$	$\Rightarrow$	$[4, 4, 1]$
$(c_1, c_4, 0)$	$\Rightarrow$	$[7, 7, 1], [9, 9, 2]$
$(c_2, c_4, -1)$	$\Rightarrow$	$[12, 12, 2], [14, 14, 3]$
$\cdots$		$\cdots$

Above is a part of the counting lists for the tree structure in Figure 4. For instance, link $(c_2, c_4, -1)$ contains two nodes, which are created during the insertion of x^2 and z^2 (relevant subsequences of x and z in Table 3). The two nodes represent element $x_4 - x_2 = -1$ and $z_4 - z_2 = -1$ in sequence x^2 and z^2 respectively. We summarize the process of building the counting tree in Algorithm 1.

Thus, our counting tree is composed of two structures, the tree and the counting lists. We observe the following properties of the counting tree:

1. For any two nodes x, y labeled $(ID_x^\vdash, ID_x^\dashv, Count_x)$ and $(ID_y^\vdash, ID_y^\dashv, Count_y)$ respectively, node y is a descendent of node x if $ID_y^\vdash \in [ID_x^\vdash, ID_x^\dashv]$.

2. Each node appears once and only once in the counting lists.

---

[2]If list (c_i, c_j, v) is empty, we make $(ID^\vdash, ID^\dashv, Count)$ the first element of the list.

3. Nodes in any counting list are in ascending order of their $ID^\vdash$.

The proof of the above properties is straightforward and we omit it here. These properties are essential to finding the dense patterns efficiently (Section 3.3).

Input: $\mathcal{D}$: a dataset in multidimensional space $\mathcal{A}$
 $\quad\quad\quad$ ξ: minimal pattern length (dimensionality)

Output: F: a counting tree

$F \leftarrow$ empty tree;
for *all objects* $x \in \mathcal{D}$ **do**
$\quad$ $i \leftarrow 1$;
$\quad$ **while** $i < |\mathcal{A}| - \xi + 1$ **do**
$\quad\quad$ insert x^i into F;
$\quad\quad$ $i \leftarrow i + 1$;

make a depth-first traversal of F;
for *each node s encountered in the traversal* **do**
$\quad$ let s represents sequence element $x_j - x_i = v$;
$\quad$ label node s by $[id_s^\vdash, id_s^\dashv, count]$;
$\quad$ $lcnt \leftarrow count$ of the last element in list (c_i, c_j, v),
$\quad$ or 0 if (c_i, c_j, v) is empty;
$\quad$ append $[id_s^\vdash, id_s^\dashv, count + lcnt]$ to list (c_i, c_j, v);

Algorithm 1: Build the Counting Tree

3.3 Counting Pattern Occurrences

We describe SeqClus, an efficient algorithm for finding the occurrence number of a specified pattern using the counting tree structure introduced above.

Each node s in the counting tree represents a pattern p, which is embodied by the path leading from the root node to t. For instance, the node s in Figure 4 represents pattern $\langle (c_1, 0), (c_2, 1) \rangle$.

How do we find the number of occurrence of pattern p' which is one element longer than p? That is,

$$ p' = \langle \underbrace{(c_i, v_i), \cdots, (c_j, v_j)}_{p}, (c_k, v) \rangle. $$

The counting tree structure makes this operation very easy. First, we only need to look for nodes in counting list (c_i, c_k, v), since all nodes of $x_k - x_i = v$ are in that list. Second, we are only interested in nodes that are under node s, because only those nodes satisfy pattern p, a prefix of p'. Assuming s is labeled $(ID_s^\vdash, ID_s^\dashv, count)$, we know s's descendent nodes are in the range of $[ID_s^\vdash, ID_s^\dashv]$. According to the counting properties, elements in any counting list are in ascending order of their $ID^\vdash$ values, which means we can binary-search the list. Finally, assume list (c_i, c_k, v) contain the following nodes:

$$ \cdots, (\_, \_, cnt_u), \underbrace{(id_v^\vdash, id_v^\dashv, cnt_v), \cdots, (id_w^\vdash, id_w^\dashv, cnt_w)}_{[ID_s^\vdash, ID_s^\dashv]}, \cdots $$

Then, we know all together there are $cnt_w - cnt_u$ objects[3] that satisfy pattern p'.

We denote the above process by $count(r, c_k, v)$, where r is a range, and in this case $r = [ID_s^\vdash, ID_s^\dashv]$. If, however, we are looking for patterns even longer than p', then instead of returning $cnt_w - cnt_u$, we shall continue the search. Let L denote the list of the sub-ranges represented by the nodes within range $[ID_s^\vdash, ID_s^\dashv]$ in list (c_i, c_k, v), that is,

$$ L = \{[id_v^\vdash, id_v^\dashv], \cdots, [id_w^\vdash, id_w^\dashv]\} $$

Then, we repeat the above process for each range in L, and the final count comes to

$$ \sum_{r \in L} count(r, c, v) $$

where (c, v) is the next element following p'.

We summarize the counting process described above in Algorithm 2.

Input: $\mathcal{Q}$: a query pattern on dataset $\mathcal{D}$
 $\quad\quad\quad$ F: the counting tree of $\mathcal{D}$

Output: number of occurrences of $\mathcal{Q}$ in $\mathcal{D}$

assume $\mathcal{Q} = \langle (q_1, 0), (q_2, v_2), \cdots, (q_j, v_j), \cdots \rangle$;
$(r, cnt) \leftarrow count(Universe, q_1, 0)$;
return $countPattern(r, 2)$;

Function $countPattern(r, j)$
the jth element of $\mathcal{Q}$ is (q_j, v_j);
$(L, cnt) \leftarrow count(r, q_j, v_j)$;
if $j = |\mathcal{Q}|$ **then**
$\quad$ **return** cnt;
else
$\quad$ **return** $\sum_{r' \in L} countPattern(r', j + 1)$
end

Function $count(r, c, v)$
$cl \leftarrow$ the counting list for (q_1, c, v);
perform range query r on cl and assume cl contain the following elements:
$\cdots, (\_, \_, cnt'), \underbrace{(id_j^\vdash, id_j^\dashv, cnt_j), \cdots, (id_k^\vdash, id_k^\dashv, cnt_k)}_{r}, \cdots$

return (L, cnt) where:
$cnt = cnt_k - cnt'$;
$L = \{[id_j^\vdash, id_j^\dashv], \cdots, [id_k^\vdash, id_k^\dashv]\}$;

Algorithm 2: Algorithm count()

3.4 Clustering

The counting algorithm in Section 3.3 finds the number of occurrences of a specified pattern, or the density of the

---

[3]or just cnt_w objects if $id_v^\vdash$ is the first element of the list.

cells in the pattern grids of a given subspace (Figure 3). We can then use a density and grid based clustering algorithm to group the dense cells together.

We start with patterns containing only two columns (in a 2-dimensional subspace), and grow the patterns by adding new columns into them. During this process, patterns that correspond to no more than `minRows` objects are pruned, as introducing new columns into the pattern will only reduce the number of objects.
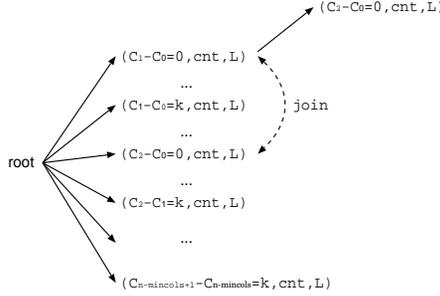
Figure 5: The Cluster Tree

Figure 5 shows a tree structure for growing the clusters. Each node t in the tree is a triple (`item`, `count`, `range-list`). The `items` in the nodes along the path from the root node to node t constitutes the pattern represented by t. For instance, the node in the 3rd level in Figure 5 represents $\langle (c_0, 0), (c_1, 0), (c_2, 0) \rangle$, a pattern in a 3-dimensional space. The value `count` in the triple represents the number of occurrences of the pattern in the dataset, and `range-list` is the list of ranges of the IDs of those objects. Both `count` and `range-list` are computed by the count() routine in Algorithm 2.

First of all, we count the occurrences of all patterns containing 2 columns, and insert them under the root node if they are frequent (count $\geq$ `minRows`). Note there is no need to consider all the columns. As any $c_i - c_j = v$ to be the first item in a pattern with at least `minCols` columns, c_i must be less than $c_{n-\texttt{minCols}+1}$ and c_j must be less than $c_{n-\texttt{minCols}}$.

In the second step, for each node p on the current level, we join p with its *eligible* nodes to derive nodes on the next level. A node q is node p's eligible nodes if it satisfies the following criteria:

- q is on the same level as p;
- if p denotes item $a - b = v$ and q denotes $c - d = v'$, then $a \prec c$, $b = d$.

Besides p's eligible nodes, we also join p with item in the form of $c_{n-\texttt{minCols}+k} - b = v$, since column $c_{n-\texttt{minCols}+k}$ does not appear in levels less than k.

The join operation is easy to perform. Assume p, represented by triple (`a - b = v`, `count`, `range-list`), is to be joined with item $c - b = v'$, we simply compute $count(r, c, v')$ for each range r in `range-list`. If the sum of the returned counts is larger than `minRows`, then we

insert a new node (`c - b = v'`, `count'`, `range-list'`) under p, where count' is the sum of the returned counts, and `range-list'` is the union of all the ranges returned by count(). Algorithm 3 summarizes the clustering process described above.

4   Experiments

We implement the algorithms in C on a Linux machine with a 700 MHz CPU and 256 MB main memory. We tested it on both synthetic and real life data sets.

Input: `minCols`: dimensionality threshold
         `minRows`: cluster size threshold
         F: tree structure for $\mathcal{D}$
Output: clusters of objects in $\mathcal{D}$

$T \leftarrow$ create root note of tree;
$Queue \leftarrow \emptyset$;
for $i = 1$ *to* $|\mathcal{A}|$-`minCols` **do**
   $(cnt, L) \leftarrow count(NULL, C_i, 0)$;
   if $cnt \geq$ `minCols` **then**
      insert $(c_i, 0, cnt, L)$ under T and into $Queue$;
   end
end
while $Queue \neq \emptyset$ **do**
   remove the 1st element x from $Queue$;
   assume $x = (c_i, v, cnt, L)$;
   join x with *eligible* node $y = (c_j, v', cnt', L')$;
   $(cnt'', L'') \leftarrow count(L, C_j, v)$;
   if $cnt'' \geq$`minRows` **then**
      Insert (c_j, v'', cnt'', L'') under x and into $Queue$;
   end
end
for *each leaf node x of the tree* **do**
   assume $x = (c_i, v, cnt, L)$;
   columns $\leftarrow$ path from root to x;
   objects $\leftarrow$ findAll(L);
   return cluster {columns,objects};
end

Algorithm 3: Clustering Algorithm

4.1   Data Sets

We generate synthetic datasets in tabular and sequential forms. For real life datasets, we use time-stamped event sequences generated by a production network (sequential data), and DNA micro-arrays of yeast and mouse gene expressions under various conditions (tabular data).

SYNTHETIC DATA   We generate synthetic data sets in tabular forms. Initially, the table is filled with random values ranging from 0 to 300, and then we embed a fixed number of clusters in the raw data. The clusters embedded can also have varying quality. We embed *perfect* clusters in the matrix, i.e., the distance between any two objects in the embedded cluster is 0 (i.e., $\delta = 0$). We

also embed clusters whose distance threshold among the objects is $\delta = 2, 4, 6, \cdots$. We also generate synthetic sequential datasets in the form of $\cdots$ (`id,timestamp`) $\cdots$, where instead of embedding clusters, we simply model the sequences by probabilistic distributions. Here, the `ids` are randomly generated; however, the occurrence rate of different `ids` follows either a uniform or a Zipf distribution. We generate ascending `timestamps` in such a way that the number of elements in a unit window follows either uniform or Poisson distribution.

GENE EXPRESSION DATA   Gene expression data are presented as a matrix. The yeast micro-array [18] can be converted to a weighted-sequence of 49,028 elements (2,884 genes under 17 conditions). The expression levels of the yeast genes (after transformation) range from 0-600, and they are discretized into 40 bins. The mouse cDNA array is 535,766 in size (10,934 genes under 49 conditions) and it is pre-processed in the same way.

EVENT MANAGEMENT DATA   The data sets we use are taken from a production computer network at a financial service company.   NETVIEW [16] has six attributes: `Timestamp`, `EventType`, `Host`, `Severity`, `Interestingness`, and `DayOfWeek`. We are concerned with attribute `Timestamp` and `EventType`, which has 241 distinctive values. TEC [16] has attributes `Timestamp`, `EventType`, `Source`, `Severity`, `Host`, and `DayOfYear`. In TEC, there are 75 distinctive values of `EventType` and 16 distinctive types of `Source`. It is often interesting to differentiate same type of events from different sources, and this is realized by combining `EventType` and `Source` to produce $75 \times 16 = 1200$ symbols.

4.2   Performance Analysis

We evaluate the scalability of the clustering algorithm on synthetic tabular datasets and compare it with pCluster [21]. The number of objects in the dataset increases from 1,000 to 100,000, and the number of columns from 20 to 120. The results presented in Figure 6 are average response times obtained from a set of 10 synthetic data.

Data sets used for Figure 6(a) are generated with number of columns fixed at 30. We embed a total of 10 perfect clusters ($\delta = 0$) in the data. The minimal number of columns of the embedded cluster is 6, and the minimal number of rows is set to $0.01N$, where N is the number of rows of the synthetic data.

The pCluster algorithm is invoked with `minCols`= 5, `minRows`= $0.01N$, and $\delta = 3$, and the SeqClus algorithm is invoked with $\delta = 3$. Figure 6(a) shows that there is almost a linear relationship between the time and the data size for the SeqClus algorithm. The pCluster algorithm, on the other hand, is not scalable, and it can only handle datasets with size in the range of thousands.

For Figure 6(b), we increase the dimensionality of the synthetic datasets from 20 to 120. Each embedded cluster is in subspace whose dimensionality is at
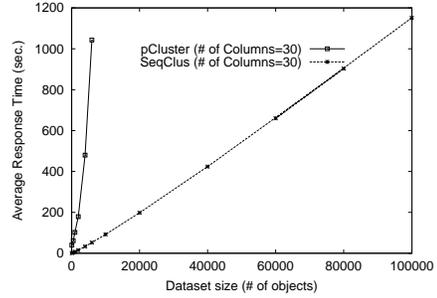
(a) Scalability with the # of rows in data sets
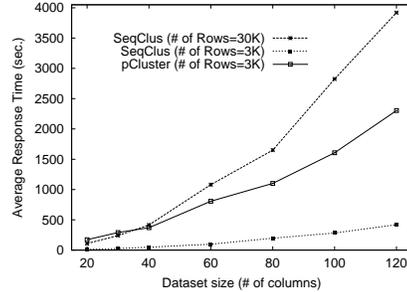
(b) Scalability with the # of columns in data sets

Figure 6: Performance Study: scalability.

least $0.02C$, where C is the number of columns of the data set. The pCluster algorithm is invoked with $\delta = 3$, `minCols`= $0.02C$, and `minRows`= 30. The curve of SeqClus exhibits quadratic behavior. However, it shows that, with increasing dimensionality, SeqClus can almost handle datasets of size an order of magnitude larger than pCluster (30K vs. 3K). We were unable to get the performance result of pCluster on datasets of 30K objects.
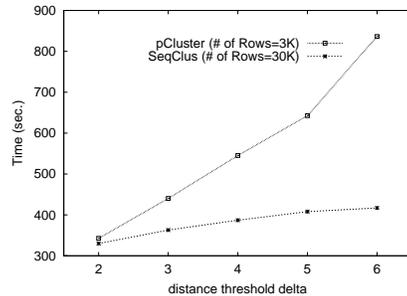
Figure 7: Time vs. distance threshold δ

Next we study the impact of the *quality* of the embedded clusters on the performance of the clustering algorithms. We generate synthetic datasets containing 3K/30K objects, 30 columns with 30 embedded clusters (each on average contains 30 objects, and the clusters are in subspace whose dimensionality is 8 on average). Within each cluster, the maximum distance (under the pCluster model) between any two objects ranges from $\delta = 2$ to $\delta = 6$. Figure 7 shows that, while the performance of the pCluster algorithm degrades with the increase of δ, the SeqClus algorithm is more robust under

8

this situation. The reason is because much of the computation of SeqClus is performed on the counting tree, which provides a compact summary of the dense patterns in the dataset, while for pCluster, a higher δ value has a direct, negative impact on its pruning effect [21].
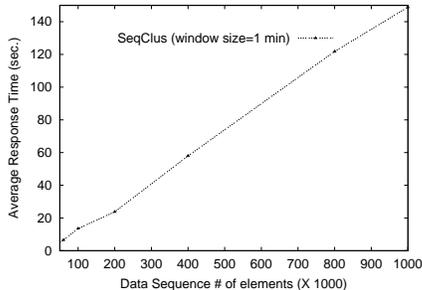
Figure 8: Scalability on sequential dataset

We also study clustering performance on timestamped sequential datasets. The dataset in use is in the form of $\cdots$ (id,timestamp) $\cdots$, where every minute contains on average 10 ids (uniform distribution). We place a sliding window of size 1 minute on the sequence, and create a counting tree for the subsequences inside the windows. The scalability result is shown in Figure 8. We also tried different distributions of id and timestamp, but did not observe significant differences in performance.

4.3 Cluster Analysis

We report the clusters found in real life datasets. Table 4 shows the number of clusters found by the pCluster and SeqClus algorithm in the raw Yeast micro-array dataset.

δ	minCols	minRows	# of clusters	
			pCluster	SeqClus
0	9	30	5	5
0	7	50	11	13
0	5	30	9370	11537

Table 4: Clusters found in the Yeast dataset

For minCols= 9 and minRows= 30, the two algorithms found the same clusters. But in general, using the same parameters, SeqClus produces more clusters. This is because the similarity measure used in the pCluster model is more restrictive. We find that the objects (genes) in those clusters overlooked by the pCluster algorithm but discovered by the SeqClus method exhibit easily perceptible coherent patterns. For instance, the genes in Figure 9 shows a coherent pattern in the specified subspace, and this subspace cluster is discovered by SeqClus but not by pCluster. This indicates the relaxation of the similarity model not only improves the performance but also provides extra insight in understanding the data.
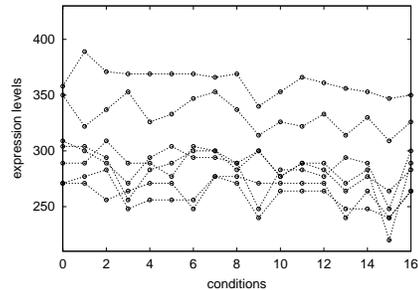
Figure 9: A cluster in subspace $\{2,3,4,5,7,8,10,11,12,13,14,15,16\}$.

The SeqClus algorithm works directly on both tabular and sequential datasets. Table 5 shows event sequence clusters found in the NETVIEW dataset [16]. We apply the algorithm on 10 days' worth of event logs (around 41M bytes) of the production computer network.

δ	# events	# sequences	SeqClus
2 sec	10	500	31
4 sec	8	400	143
6 sec	6	300	2276

Table 5: Clusters found in NETVIEW

5 Related Work

The study of clustering based on pattern similarity is related to previous work on subspace clustering. Many recent studies [2, 3, 4, 7, 12] focus on mining subspace clusters embedded in high-dimensional spaces.

Still, strong correlations may exist among a set of objects even if they are far apart from each other as measured by distance functions (such as Euclidean) used frequently in traditional clustering algorithms. Many scientific projects collect data in the form of Figure 1, and it is essential to identify clusters of objects that manifest coherent patterns. A variety of applications, including DNA microarray analysis, E-commerce collaborative filtering, will benefit from fast algorithms that can capture such patterns. Cheng et al [8] proposed the bicluster model, which captures the coherence of genes and conditions in a sub-matrix of a DNA micro-array.

In this paper, we show that clustering by pattern similarity is closely related to the problem of subsequence matching. There has been much research on string indexing and substring matching. For instance, a suffix tree [14] is a very useful data structure that embodies a compact index to all the distinct, non-empty substrings of a given string. Suffix arrays [13] and PAT-arrays [11] also provide fast searches on text databases. Similarity based subsequence matching [10, 17] has been a research focus for applications such as time series databases.

6 Conclusion

Clustering by pattern similarity is an interesting and challenging problem. The computational complexity problem of subspace clustering is further aggravated by the fact that we are concerned with patterns of rise and fall instead of value similarity. The task of clustering by pattern similarity can be converted into a traditional subspace clustering problem by (i) creating a new dimension ij for every two dimension i and j of any object x, and set x_{ij}, the value of the new dimension, to $x_i - x_j$; or (ii) creating $|\mathcal{A}|$ copies ($\mathcal{A}$ is the entire dimension set) of the original dataset, where x_k, the value of x on the kth dimension in the ith copy is changed to $x_k - x_i$, for $k \in \mathcal{A}$. For both cases, we need to find subspace clusters in the transformed dataset, which is $|\mathcal{A}|$ times larger. These methods are apparently not feasible for datasets in high dimensional spaces. They also cannot be applied to sequential datasets, for instance, in event management systems where millions of timestamped events are generated on a daily basis. In this paper, we introduced a sequence based similarity measure to model pattern similarity. We proposed an efficient implementation, the counting tree, which is based on the suffix tree structure. Experimental results show that the SeqClus algorithm achieves an order of magnitude speedup over the current best algorithm pCluster. The new model also enables us to identify clusters overlooked by previous methods such as the pCluster model. Furthermore, the sequence model is natural to be applied on sequential data directly.

References

[1] Alison Abbott. Bioinformatics institute plans public database for gene expression data. *Nature*, 398:646, 1999.

[2] C. C. Aggarwal, C. Procopiuc, J. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD*, 1999.

[3] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, pages 70–81, 2000.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Authomatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

[5] Alvis Brazma, Alan Robinson, Graham Cameron, and Michael Ashburner. One-stop shop for microarray data. *Nature*, 403:699–700, 2000.

[6] P. O. Brown and D. Botstein. Exploring the new world of the genome with DNA microarrays. *Nature Genetics*, 21:33–37, 1999.

[7] C. H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *SIGKDD*, pages 84–93, 1999.

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

[9] P. D'haeseleer, S. Liang, and R. Somogyi. Gene expression analysis and genetic network modeling. In *Pacific Symposium on Biocomputing*, 1999.

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[11] G. Gonnet, R. Baeza-Yates, and T. Snider. New indices for text: Pat trees and pat arrays. In *Information Retrieval: Data Structures and Algorithms*, pages 335–349. Prentice Hall, 1992.

[12] H. V. Jagadish, Jason Madar, and Raymond Ng. Semantic compression and pattern extraction with fascicles. In *VLDB*, pages 186–196, 1999.

[13] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal On Computing*, pages 935–948, 1993.

[14] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.

[15] Jian Pei, Xiaoling Zhang, Moonjung Cho, Haixun Wang, and Philip S. Yu. Maple: A fast algorithm for maximal pattern-based clustering. In *ICDM*, 2003.

[16] Chang-Shing Perng, Haixun Wang, Sheng Ma, and Joseph L Hellerstein. A framework for exploring mining spaces with multiple attributes. In *ICDM*, 2001.

[17] Chang-Shing Perng, Haixun Wang, Sylvia R. Zhang, and D. Stott Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, pages 33–42, 2000.

[18] S. Tavazoie, J. Hughes, M. Campbell, R. Cho, and G. Church. Yeast micro data set. In *http://arep.med.harvard.edu/biclustering/yeast.matrix*, 2000.

[19] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. ViST: A dynamic index method for querying XML data by tree structures. In *SIGMOD*, 2003.

[20] Haixun Wang, Chang-Shing Perng, Wei Fan, Sanghyun Park, and Philip S. Yu. Indexing weighted sequences in large databases. In *ICDE*, 2003.

[21] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD*, 2002.

[22] Jiong Yang, Wei Wang, Haixun Wang, and Philip S Yu. δ-clusters: Capturing subspace correlation in a large data set. In *ICDE*, pages 517–528, 2002.