Pattern-based Similarity Search for Microarray Data

Haixun Wang
IBM T. J. Watson Research
Hawthorne, NY 10532
haixun@us.ibm.com

Jian Pei
Simon Fraser University
Canada
jpei@cs.sfu.ca

Philip S. Yu
IBM T. J. Watson Research
Hawthorne, NY 10532
psyu@us.ibm.com

ABSTRACT

One fundamental task in near-neighbor search as well as other similarity matching efforts is to find a distance function that can efficiently quantify the similarity between two objects in a meaningful way. In DNA microarray analysis, the expression levels of two closely related genes may rise and fall synchronously in response to a set of experimental stimuli. Although the magnitude of their expression levels may not be close, the patterns they exhibit can be very similar. Unfortunately, none of the conventional distance metrics such as the L_p norm can model this similarity effectively. In this paper, we study the near-neighbor search problem based on this new type of similarity. We propose to measure the distance between two genes by subspace pattern similarity, i.e., whether they exhibit a synchronous pattern of rise and fall on a subset of dimensions. We then present an efficient algorithm for subspace near-neighbor search based on pattern similarity distance, and we perform tests on various data sets to show its effectiveness.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Pattern analysis*

Keywords

pattern recognition, near neighbor, distance function

1. INTRODUCTION

Given a distance function $dist(\cdot,\cdot)$ that measures the similarity between two objects, a query object q's near-neighbors within a given tolerance radius r in a database $\mathcal{D}$ is defined as:

$$\mathcal{NN}(q,r) = \{p|\, p \in \mathcal{D}, dist(q,p) \leq r\} \qquad (1)$$

The distance function $dist(\cdot,\cdot)$ has a direct impact on the efficiency of the search of near-neighbors [3]. More importantly, it also determines the meaning of similarity and the meaning of the near-neighbor search.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

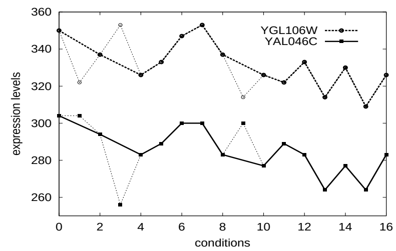


Figure 1: Similarity between two yeast genes

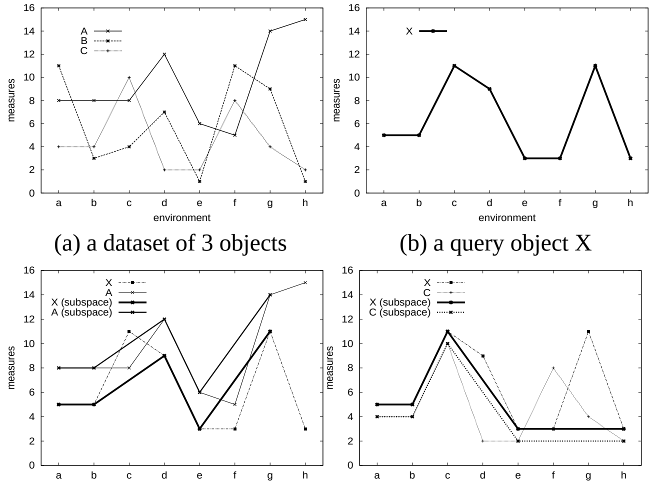
Applications. In this paper, we address a new type of similarity that cannot be effectively captured by conventional distance metric such as the L_p norm. As a motivating example, in Figure 1, we show the expression levels of two Yeast genes under 17 different external conditions. It is clear that the two genes manifest similarity under a subset of conditions (linked in thick lines).

In many scientific experiments, we measure objects in different environments. Figure 2(a) is a dataset that contain the measurements of 3 objects in 8 different environments. Now, given a new object X whose measurements are shown in Figure 2(b), we want to find X's near neighbors in the dataset. Figure 2(c) and (d) show potential near-neighbors of object X. In Figure 2(c), the values of object X and A rise and fall coherently under conditions $\{a, b, d, e, g\}$. Figure 2(d) reveals, in much the same way, the similarity of X and C under $\{a, b, c, e, h\}$.

Finding near neighbors based on subspace pattern similarity is important to many applications including DNA microarray analysis [1, 8, 7]. A DNA microarray is a two dimensional matrix where entry d_{ij} represents the expression level of gene i in sample j. Investigations show that more often than not, several genes contribute to a disease, which motivates researchers to identify genes whose expression levels rise and fall synchronously under a subset of conditions, that is, whether they exhibit fluctuation of a similar shape when conditions change.

Problems. Assume we are given a new gene for which we do not know in which conditions it might manifest coherent patterns with other genes. This new gene might be related to any gene in the database as long as both of them exhibit a pattern in some subspace. The dimensionality of the subspace is often an indicator of the degree of their closeness, that is, the more columns the pattern spans, the closer the relationship between the two genes.

EXAMPLE 1 (NEAR-NEIGHBOR SEARCH IN ANY SUBSPACES). *Given a gene q, and a dimensionality threshold r, find all genes whose expression levels manifest coherent patterns with those of q*



(a) a dataset of 3 objects (b) a query object X

(c) X and A (d) X and C

Figure 2: What is a near neighbor?

in any subspace S, where $|S| \geq r$.

Our Contributions. We introduce a new measure to capture pattern-based similarity exhibited by objects in subspaces. With the new distance measure, we extend the concept of near-neighbor to the realm of pattern-based similarity, which often carries significant meanings. We also propose a novel method to perform near-neighbor search by pattern similarity. Experiments show that our method is effective and efficient, and it outperforms alternative algorithms (based on an adaptation of the R-Tree index) by an order of magnitude.

2. PATTERN DISTANCE

Let u, v be two objects in dataset $\mathcal{D}$. How can we measure their pattern-based similarity in a given subspace, say $\mathcal{S} = \{a, b, c, d, e\}$?

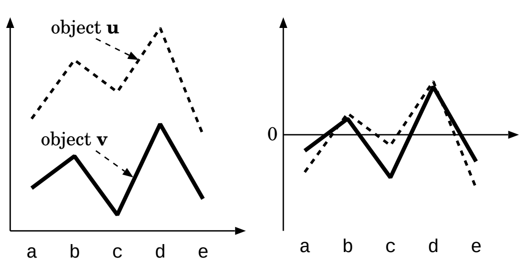


Figure 3: pattern in subspace $\mathcal{S} = \{a, b, c, d, e\}$

A straightforward way is to *normalize* both objects in subspace $\mathcal{S}$ (Figure 3) by shifting u and v by an amount of $\overline{u_s}$ and $\overline{v_s}$ respectively, where $\overline{u_s}$ ($\overline{v_s}$) is the average coordinate value of u (v) in subspace $\mathcal{S}$. After normalization, we can check whether u and v exhibit a pattern of good quality in subspace $\mathcal{S}$:

DEFINITION 1 (COHERENT PATTERN). *Objects u, $v \in \mathcal{D}$ exhibit a coherent pattern in subspace $S \subseteq \mathcal{A}$ if*

$$d_{\mathcal{S}}(u, v) = \max_{i \in \mathcal{S}} |(u_i - \overline{u_s}) - (v_i - \overline{v_s})| \leq \epsilon \quad (2)$$

where $\overline{u_s} = \frac{1}{|S|}\sum_{i \in S} u_i$, $\overline{v_s} = \frac{1}{|S|}\sum_{i \in S} v_i$ are average coordinate values of u and v in subspace S, and $\epsilon \geq 0$.

The above definition, although intuitive, may not be applicable or effective for near-neighbor search in arbitrary subspaces. Near-neighbor search queries often rely on index structures to speed up the search process. The definition of the coherent pattern (Eq 2) uses not only coordinate values (i.e., u_i, v_i) but also average coordinate values in subspaces (i.e., $\overline{u_s}, \overline{v_s}$). It is unrealistic, however, to index average values for each of the $2^{|\mathcal{A}|}$ subsets.

To avoid the curse of dimensionality, we relax Definition 1 by eliminating the need of computing average values in Eq 2. Instead, we use the coordinate values of *any* column $k \in \mathcal{S}$ as the base for comparison. Given a subspace $\mathcal{S}$ and any column $k \in \mathcal{S}$, we define:

$$d_{k,\mathcal{S}}(u, v) = \max_{i \in \mathcal{S}} |(u_i - u_k) - (v_i - v_k)| \quad (3)$$

However, the choice of column k may be questionable: does an arbitrary k affect our ability in capturing pattern similarity? The following property relieves this concern.

THEOREM 1. *If there exists $k \in \mathcal{S}$ such that $d_{k,\mathcal{S}}(u, v) \leq \epsilon$, then we have:*

$$\forall i \in S \;\; d_{i,\mathcal{S}}(u, v) \leq 2\epsilon \qquad and \qquad d_{\mathcal{S}}(u, v) < 2\epsilon$$

PROOF. (sketch) Note:

$$\begin{aligned} |(u_j - u_i) - (v_j - v_i)| \quad \leq \quad & |(u_j - v_j) - (u_k - v_k)| + \\ & |(u_k - v_k) - (u_i - v_i)|, \end{aligned}$$

and

$$|(u_i - \overline{u_{\mathcal{S}}}) - (v_i - \overline{v_{\mathcal{S}}})| \leq \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{S}} |(u_i - u_j) - (v_i - v_j)|.$$

□

Not only the difference among base columns is limited, Theorem 1 also shows that, the difference between using Eq 2 and Eq 3 is bounded by a factor of 2 in terms of the pattern quality. In the same light, we can show that if u and v exhibit an coherent pattern in subspace $\mathcal{S}$, then $\forall k \in \mathcal{S}$, we have $d_{k,\mathcal{S}}(u, v) \leq 2\epsilon$. Thus, in order to find all coherent pattern, we can use $d_{k,\mathcal{S}}(u, v) \leq 2\epsilon$ as the criteria and then prune the results, since Eq 3 is much less costly to compute.

In order to find patterns defined by a consistent measure, we fix the base column k for any subspace $\mathcal{S} \subseteq \mathcal{A}$. We assume there is a total order among the dimensions in $\mathcal{A}$. Given a subspace $\mathcal{S}$, we use its least dimension in terms of the total order as the base column. Now we arrive at the definition of ϵ-pattern that induces an efficient implementation.

DEFINITION 2 (ϵ-PATTERN). *Objects u, $v \in \mathcal{D}$ exhibit an ϵ-pattern in subspace $S \subseteq \mathcal{A}$ if*

$$d_{k,\mathcal{S}}(u, v) \leq \epsilon$$

where k is the least dimension in $\mathcal{S}$ and $\epsilon \geq 0$.

The ϵ-pattern definition focuses on pattern similarity in a given subspace. How to measure similarity between two objects when no subspace is specified? Usually, we do not care over which subspace two objects exhibit a similar pattern, but rather, how many dimensions the pattern spans. Thus, the dimensionality of the subspace can be used as an indicator of the degree of the similarity.

DEFINITION 3 (SUBSPACE PATTERN SIMILARITY). *Given two objects u, $v \in \mathcal{D}$ and some $\epsilon \geq 0$, we say that the similarity between u and v is r, or*

$$s(u,v) = r$$

if r is the maximum dimensionality of all subspaces $S \subseteq \mathcal{A}$ where u and v exhibit an ϵ-pattern.

Thus, two objects that exhibit an ϵ-pattern in the entire space $\mathcal{A}$ will have the largest similarity of $|\mathcal{A}|$. The distance between the two objects is reversely proportional to their similarity. For instance, we can define $dist(u,v) = 1/s(u,v)$. Note that the distance defined above is non-metric in that it does not satisfy the triangular inequality. One object can share ϵ-patterns with two other objects in different subspaces, and the sum of the distances to the two objects might be smaller than the distance between the two objects, which may not share synchronous patterns in any subspace.

Problem Statement. We define the following problem of near neighbor search. Given an object q, a tolerance radius r, find $\mathcal{NN}(q,r)$ in dataset $\mathcal{D}$:

$$\mathcal{NN}(q,r) = \{u \in \mathcal{D} \mid s(q,u) \geq r\} \quad (4)$$

3. NEAR-NEIGHBOR SEARCH BY SUBSPACE PATTERN SIMILARITY

In this section, we propose a new index structure called *PS-Index (pattern similarity index)* to support near-neighbor search in pattern distance space.

3.1 Building a Trie

Given a dataset $\mathcal{D}$ in space $\mathcal{A} = \{c_1, c_2, ..., c_n\}$, where $c_1 \prec c_2 \prec \cdots \prec c_n$ is a total order on attributes, we represent each object $u \in \mathcal{D}$ as a sequence of (column, value) pairs, that is:

$$u = (c_1, u_1), (c_2, u_2), ..., (c_n, u_n)$$

An *aligned suffix* of u is defined as:

$$f(u,i) = (c_i, 0), (c_{i+1}, u_{i+1} - u_i), ..., (c_k, u_k - u_i) \quad (5)$$

We use an example to demonstrate the data sequentializing process.

EXAMPLE 2. *Let database $\mathcal{D}$ be composed of the following object defined in space $\mathcal{A} = \{c_1, c_2, c_3, c_4, c_5\}$.*

obj	c_1	c_2	c_3	c_4	c_5
#1	*3*	*0*	*4*	*2*	*0*

We derive all aligned suffices of length ≥ 2 of the object, and insert them into a trie. Figure 4 demonstrates the insertion of the following sequence:

$$f(\#1, 1) = (c_1, 0), (c_2, -3), (c_3, 1), (c_4, -1), (c_5, -3)$$

Each leaf node n in the trie maintains an *object list*, L_n. Assuming the insertion of $f(\#1, 1)$ leads to node x, which is under arc $(e, -3)$, we append 1 (object #1), to object list L_x.

3.2 Building PS-Index over a Trie

The trie enables us to find near-neighbors of a query object $q = (c_1, v_1), ..., (c_n, v_n)$ in a given subspace S, provided S is defined by a set of *consecutive* columns, i.e., $S = \{c_i, c_{i+1}, ..., c_{i+k}\}$. The PS-index, described below, allows us to 'jump' directly from a column c_j to any column c_k, where $k > j$.

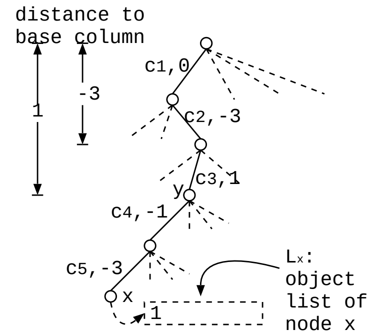


Figure 4: Insertion of sequence $f(\#, 1)$.

We use the following two steps to build the PS-index on top of a trie. First, after all sequences are inserted, we assign to each node x a pair of labels, $\langle n_x, s_x \rangle$, where n_x is the prefix-order of node x in the trie (starting from 0, which is assigned to the root node), and s_x is the number of x's descendent nodes.

Next, we store nodes into buffers. For each unique edge $(col, dist)$ in the trie[1], we create a buffer. Nodes are appended to the buffers during a depth-first walk of the trie. When we encounter a node x under edge $(col, dist)$, we append x's label $\langle n_x, s_x \rangle$ to the buffer of $(col, dist)$. From the definition of base-column aligned suffixes, it is clear that a buffer is composed of nodes that have the same distance from their base columns (root node).

The labeling scheme and the node buffers have the following property.

THEOREM 2 (PS-INDEX PROPERTY).

1. *If node x and y are labeled $\langle n_x, s_x \rangle$ and $\langle n_y, s_y \rangle$ respectively, and $n_x < n_y \leq n_x + s_x$, then y is a descendent node of x;*
2. *nodes in any link are ordered by their prefix-order number; and*
3. *if a link contains nodes $u \ldots v \ldots w$ (in that order), and u and w have a common ancestor x, then x is also v's ancestor.*

PROOF. 1) and 2) are due to the labeling scheme which is based on depth-first traversal. For 3), note that if nodes $u, ..., v, ..., w$ are in a link, and u, v are descendents of x, we have $n_x < n_u < n_v < n_w \leq n_x + s_x$, which means v is also a descendent of x. $\square$

The above properties enable us to use range queries to find descendents of a given node in a given link.

Algorithm 1 summarizes the index construction procedure. The time complexity of building the PS-index is $O(|\mathcal{D}||\mathcal{A}|)$. The Ukkonen algorithm [6] builds suffix tree in linear time. The construction of the trie for pattern-similarity indexing is less time consuming because the length of the indexed subsequences is constrained by $|\mathcal{A}|$. Thus, it can be constructed by a brute-force algorithm [4] in linear time. The space taken by the PS-Index is linearly proportional to the data size. Since each node appears once and only once in the pattern distance links, the total number of entries equals the total number of nodes in the trie, or $O(|\mathcal{D}||\mathcal{A}|^2)$ in the worst case (if none of the nodes are shared by any subsequences). On the other hand, there are exactly $|\mathcal{D}|(|\mathcal{A}| - 1)$ object ids stored. Thus, the space is linearly proportional to the data size $|\mathcal{D}|$.

[1] For each column col, there are at most $2\xi - 1$ unique edges, where ξ is the number of unique values of that column.

Input: $\mathcal{D}$: objects in multi-dimensional space $\mathcal{A}$
Output: PS-Index of $\mathcal{D}$

for *each* $u \in \mathcal{D}$ **do**
  insert $f(u,i), 1 \le i < |\mathcal{A}|$ into a trie; (Eq 5)
for *each node* x *encountered in a depth-first traversal of the trie* **do**
  label node x by $\langle n_x, s_x \rangle$;
  let (c,d) be the arc that points to x;
  append $\langle n_x, s_x \rangle$ to link (c,d);

Algorithm 1: Index Construction

3.3 Near-Neighbor Search

In this section, we provide an efficient solution to the 2nd problem defined in Section 2.

The Coverage Property. Each node x in the trie represents a coverage, which we denote as a range $c(x) = [n_x, n_x + s_x]$ (assuming x is labeled $\langle n_x, s_x \rangle$). Finding near-neighbors with similarity $\ge r$ boils down to finding leaf nodes whose preorder number is inside at least r ranges associated with the query object.

Let q be a query object, and $p \in \mathcal{D}$ be a near-neighbor of q (with similarity above threshold r, or $s(p,q) \ge r$). Hence, there exists a subspace S, $|S| = r$, in which p and q share a pattern. Consider $f(q,i) = (c_i, 0), \cdots, (c_k, q_k - q_i), \cdots, (c_{|\mathcal{A}|}, q_{|\mathcal{A}|} - q_i)$. Each element $(c_k, q_k - q_i)$ of $f(q,i)$ corresponds to a pattern distance link, which contains a set of nodes. Let $P(q,i)$ denote the set of all nodes that appear in the pattern distance links of the elements in $f(q,i)$, and let $P(q) = \bigcup_{i \in \mathcal{A}} P(q,i)$.

THEOREM 3. *(The Coverage Property) For any object p that shares a pattern with query object q in subspace S, there exists a set of $|S|$ nodes $\{x_1, ..., x_{|S|}\} \subseteq P(q)$, and a leaf node y that contains p ($p \in L_y$), such that $n_y \in c(x_1) \subseteq \cdots \subseteq c(x_{|S|})$, where n_y is the prefix-order of node y.*

PROOF. (Sketch) Let c_i be the *first* column of S (that is, there does not exist any $c_j \in S$ such that $j < i$). Assume the insertion of $f(p,i)$ follows the path consisting of nodes $x_i, x_{i+1}, \cdots, x_{|\mathcal{A}|}$, which leads to $c(x_{|\mathcal{A}|}) \subseteq \cdots \subseteq c(x_{i+1}) \subseteq c(x_i)$. Assume node x_j is in the list of $(c_j, p_j - p_i)$. Since p and q share pattern in S, $(c_j, p_j - p_i) = (c_j, p_j - q_i)$ holds for at least $|S|$ different columns, which means $|S|$ of the nodes in $x_i, x_{i+1}, \cdots, x_{|\mathcal{A}|}$ also appear in $P(q,i) \subset P(q)$. $\square$

The proof also shows that, to find objects that share patterns with q in subspace S, of which c_i is the *first* column, we only need to consider ranges of the objects in $P(q,i)$, instead of in the entire object set $P(q)$.

The reverse of the coverage property is also true, and can be proved under the same spirit: for any $\{x_1, \cdots, x_n\} \subseteq P(q)$ satisfying $c(x_1) \subseteq \cdots \subseteq c(x_n)$, any object $\in L_{x_1}$ is a near-neighbor of q with similarity $r \ge n$.

The Algorithm. Based on the coverage property, to find $\mathcal{NN}(q,r)$, we need to find those leaf nodes whose preorder number is inside at least r nested ranges. We perform near-neighbor search iteratively. At the ith step, we find objects that share patterns with q in subspace S, of which c_i is the first column. During that step, we only need to consider ranges of objects in $P(q,i)$.

We demonstrate the search process with an example.

EXAMPLE 3. *Given a query object,*

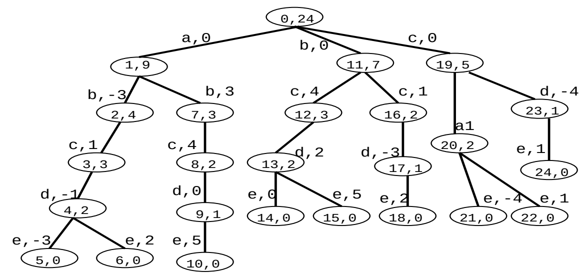
$$q = (a,1), (b,1), (c,2), (d,0), (e,3)$$

find $\mathcal{NN}(q,3)$ in $\mathcal{D}$ (Table 1).

By definition, $\forall p \in \mathcal{NN}(q,3)$, p and q must share a pattern in 3- or higher-dimensional space.

obj	a	b	c	d	e
(1)	3	0	4	2	0
(2)	4	1	5	3	6
(3)	1	4	5	1	6
(4)	0	3	4	0	5

Table 1: dataset $\mathcal{D}$ in Example 3



node	5	6	10	14	15	18	21	22	24
objs	{1}	{2}	{3,4}	{1}	{2}	{3,4}	{1}	{2}	{3,4}

Figure 5: suffix trie and object lists of dataset $\mathcal{D}$

In Figure 5, we show a labeled trie built on $\mathcal{D}$, and the object lists associated with each leaf node of the trie. For presentation simplicity, we did not include suffixes of length less than 3 in Figure 5. It does not affect the result of Example 3, which looks for patterns in 3- or higher-dimensional space.

We start with $f(q,1)$, that is, we look for patterns in subspaces that contain column a (the 1st column of $\mathcal{A}$).

$$f(q,1) = (a,0), (b,0), (c,1), (d,-1), (e,2)$$

For each element in $f(q,1)$, we consult the corresponding horizontal link and record the labels of the nodes in the horizontal link. For instance, $(a,0)$ finds one node, which is labeled $\langle 1,9 \rangle$. We record it in Figure 6. For the remaining elements of $f(q,1)$, our search is confined within that range, since we are looking for subspaces where column a is present. We consult the horizontal link of elements in $f(q,1)$ one by one. After we consult $(b,0)$, $(c,1)$, and $(d,-1)$ and record the results, we find region $[4,6]$ inside three brackets (Figure 6). It means objects in the leaf nodes whose prefix-order are in range $[4,6]$ already match the query object in a 3-dimension space. To find what those objects are, we perform a range query $[4,6]$ in the object list table shown in Figure 5, which returns object 1 and 2, and they belong to leaf node 5 and 6 respectively. The two objects share a pattern with q in 3-dimension space $\{a,c,d\}$. We repeat this process for $f(q,2)$, and so on.

Optimization. In essence, the searching process maintains a set of embedded ranges represented by brackets (Figure 6), and the goal is to find regions within r brackets. The performance of the search can be greatly improved by immediately dropping those regions from further consideration if i) all nodes inside the region

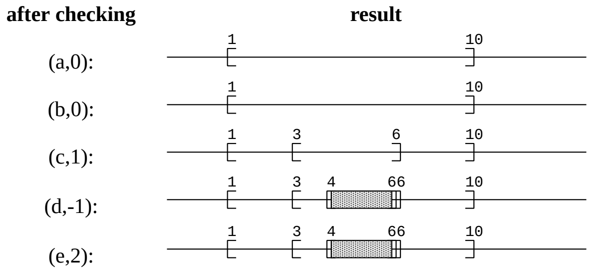


Figure 6: Embedded ranges during the search of $f(q, 1)$

already satisfy the query, or ii) no node inside the region can possibly satisfy the query. More specifically,

1. A region inside less than $r - |\mathcal{A}| + i$ brackets after the ith dimension of $\mathcal{A}$ is checked is discarded. It is easy to see that such regions will not be inside r brackets after the remaining $|\mathcal{A}| - i$ dimensions are checked.

2. If a region is already inside r brackets, we output the objects in the leaf nodes within that region, and discard the region (unless the users want the output objects ordered by their distance to the query object.)

For instance, in Figure 6, after the range of $[4, 6]$ is returned, only region $[3, 4]$ shall remain before $(e, 2)$ is checked.

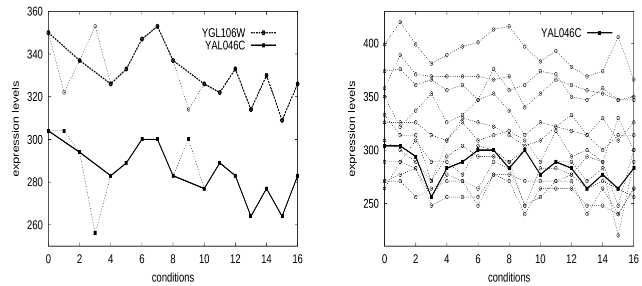
Input: $q = (c_1, v_1), \cdots, (c_n, v_n)$: a query object
r: distance threshold, ϵ: pattern tolerance
F: index file for $\mathcal{D}$
Output: $\mathcal{NN}(q, r)$

for $i = 1, ..., r + 1$ **do**
  $R \leftarrow$ the range of the (only) node in link $(c_i, 0)$;
  $j \leftarrow i + 1$;
  while $R \neq \phi$ **and** $j \leq |\mathcal{A}|$ **do**
    search link (c_j, v) for nodes inside any range of R, where $v \in [v_j - v_i - \epsilon, v_j - v_i + \epsilon]$;
    update R by adding the ranges of those nodes;
    if *a region s of R is inside $|\mathcal{A}| - r$ brackets* **then**
      output objects in L_x where $x \in s$;
      eliminate s from R;
    end
    if *a region s of R is inside less than $r - j$ brackets* **then**
      eliminate the region from s;
    end
    $j \leftarrow j + 1$;
  end
end

Algorithm 2: Near-Neighbor Search

4. EXPERIMENTS

We tested PS-Index with both synthetic and real life data setson a Linux machine with a 700 MHz CPU and 256 MB main memory. The yeast micro-array is a $2,884 \times 17$ matrix (2,884 genes under 17 conditions) [5]. The mouse cDNA array is a $10,934 \times 49$ matrix (10,934 genes under 49 conditions) [2] and it is pre-processed in the same way. We also generate synthetic data, which are random integers from a uniform distribution in the range of 1 to ξ. Let $|\mathcal{D}|$ be the number of objects in the dataset and $|\mathcal{A}|$ the number of dimensions. The total data size is $4|\mathcal{D}||\mathcal{A}|$ bytes.



(a) $s(\cdot, \cdot) \geq 13$ (b) $s(\cdot, \cdot) \geq 12$

Figure 7: Near-neighbors of YAL046C ($\epsilon = 20$)

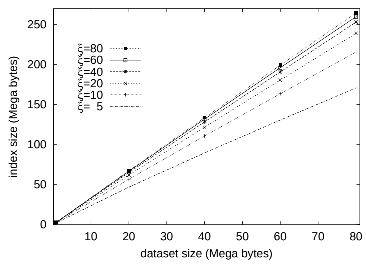
Near-Neighbor Search Results. We show results of near-neighbor search over the yeast microarray data, where genes' expression levels (of range 0 to 600 [1]) have been discretized into $\xi = 30$ bins. Assume we are interested in genes related to gene YAL046C. Let $\epsilon = 20$ (or 1 after discretization). We found one gene, YGL106W, within pattern distance 3 of gene YAL046C, i.e., YAL046C and YGL106W exhibit an ϵ-pattern in a subspace of dimensionality 14. This is illustrated by Figure 7(a), where except under conditions 1, 3, and 9 (*CH1B*, *CH2I*, and *RAT2*), the expression levels of the two genes rise and fall in sync.

Figure 7(b) shows 11 near-neighbors of YAL046C found with distance radius of 4. That is, except for 4 columns, each of the 11 genes shares an ϵ-pattern with YAL046C. It turns out that none of any two genes share ϵ-patterns with YAL046C in the same subspace. Naturally, these genes do not show up together in any subspace cluster discovered by algorithms such as bicluster [1]. Thus, subspace near-neighbor search may provide insights to understanding their interrelationship overlooked by previous techniques.

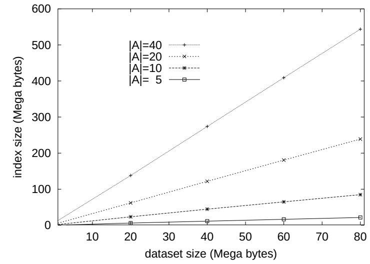
Space Analysis. The space requirement of the pattern-similarity index is linearly proportional to the data size (Figure 8). In Figure 8(a), we fix the dimensionality of the data at 20 and change ξ, the discretization granularity, from 5 to 80. It shows that ξ has little impact on the index size when the data size is small. When the data size increases, the growth of the trie slows down as each trie node is shared by more objects (this is more obvious for smaller ξ in Figure 8(a)).

In Figure 8(b) and 8(c), the discretization granularity ξ is fixed at 20, while the dimensionality of the dataset varies. The dimensionality affects the index size. With a dataset of dimensionality $|\mathcal{A}|$, the lowest similarity between two objects is 2, i.e., they do not share patterns in any subspace of dimensionality 2 or larger. However, given a query object q, our interest is in finding near-neighbors of q, that is, finding $\mathcal{NN}(q, r)$ where the similarity threshold r is high. Thus, instead of inserting each suffix of an object sequence into the trie, we insert only those suffixes of length larger than a threshold t. This enables us to find $\mathcal{NN}(q, r)$ where $r \geq t$. For a 40 Mbytes dataset of dimensionality $|\mathcal{A}|$ =80, restricting near-neighbor search within $r \geq 72$ reduces the index size by 71%.

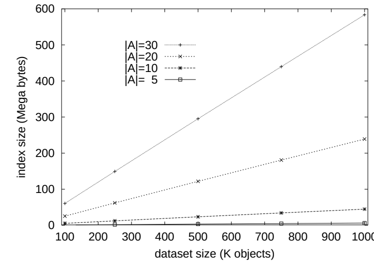
Time Analysis. We compare the algorithms presented in this paper with two alternative approaches, i) brute force linear scan, and ii) R-Tree family indices. The linear scan approach for near-neighbor search is straightforward to implement. The R-Tree, how-



(a) $|A| = 20, \xi = 5, ..., 80$

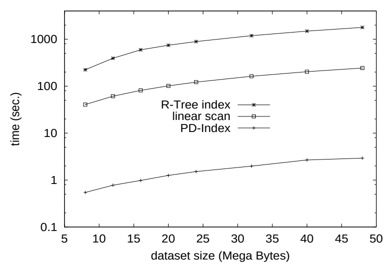


(b) varying total data size, $\xi = 20$

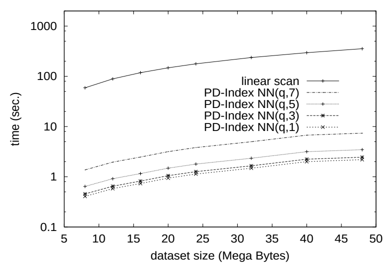


(c) varying # of objects, $\xi = 20$

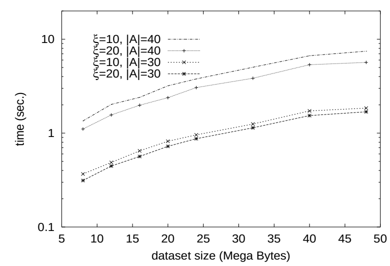
Figure 8: Index Size.



(a) Pattern matching in given subspaces



(b) Near-neighbor search in subspaces beyond given dimensionalities



(c) Impact of ξ and $|\mathcal{A}|$ in near-neighbor query $\mathcal{NN}(q, 7)$

Figure 9: Query Performance (average of 1000 runs).

ever, indexes values not patterns. To support queries based on pattern similarity, we create an extra dimension $c_{ij} = c_i - c_j$ for every two dimensions c_i and c_j. Still, R-Tree index supports only queries in given subspaces and does not support finding near-neighbors that manifest patterns in any subspace of dimensionality above a given threshold.

The query time presented in Figure 9(a) indicates that PS-Index scales much better than the two alternative approaches for pattern matching in given subspaces. The comparisons are carried out on synthetic datasets of dimensionality $|\mathcal{A}| = 40$ and discretization level $\xi = 20$. Each time, a subspace is designated by randomly selecting 4 dimensions, and random query objects are generated in the subspace. We find that the R-Tree approach is slower than brute force linear-scan for two reasons: i) the R-Tree approach degrades to linear-scan under high-dimensionality, and ii) the fact that it indexes on a much larger dataset (with $|\mathcal{A}|^2/2$ extra dimensions) means that it scans a much larger index file. In Figure 9(b), we show the results of near-neighbor search with different tolerance radiuses. PS-Index is much faster than linear-scan[2]. Still, the response time of PS-Index increases rapidly when the radius expands, as a lot more branches have to be traversed in order to find all objects satisfying the criteria. Figure 9(c) also confirms that dimensionality is a major concern in query performance.

5. CONCLUSIONS

We identify the need of finding near-neighbors under subspace pattern similarity, a new type of similarity not captured by Euclidean, Manhattan, etc., but essential to a wide range of applications, including DNA microarray analysis and e-commerce target marketing. Two objects are similar if they manifest a coherent pattern of rise and fall in an arbitrary subspace, and their degree of similarity is measured by the dimensionality of the subspace. A non-metric distance function is defined to model near-neighbor search in subspaces. We propose PS-Index, which maps objects to sequences and index them using a tree structure. Experimental results show that PS-Index achieves orders of magnitude speedup over alternative algorithms based on naive indexing and linear scan.

[2]The complexity of checking whether two objects manifest an ϵ-pattern in a subspace of dimensionality beyond a given threshold is at least $O(n \log(n))$, where $n = |\mathcal{A}|$.

6. REFERENCES

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

[2] R. Miki et al. Delineating developmental and metabolic pathways in vivo by expression profiling using the riken set of 18,816 full-length enriched mouse cDNA arrays. In *Proceedings of National Academy of Sciences*, 98, pages 2199–2204, 2001.

[3] Piotr Indyk. On approximate nearest neighbors in non-euclidean spaces. In *IEEE Symposium on Foundations of Computer Science*, pages 148–155, 1998.

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

[5] 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.

[6] E. Ukkonen. Constructing suffix-trees on-line in linear time. *Algorithms, Software, Architecture: Information Processing*, pages 484–92, 1992.

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

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