

An Index Structure for Pattern Similarity Searching in DNA Microarray Data

Haixun Wang Chang-Shing Perng Wei Fan Philip S. Yu
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
{haixun, perng, weifan, psyu}@us.ibm.com

Abstract

The DNA microarray technology is about to bring an explosion of gene expression data that may dwarf even the human sequencing projects. Researchers are motivated to identify genes whose expression levels rise and fall coherently under a set of experimental perturbances, that is, they exhibit fluctuation of a similar shape when conditions change. In this paper, we show that queries based on pattern correlations against large-scale microarray databases can be supported by the weighted-sequence model, an index structure designed for sequence matching. A weighted-sequence is a two-dimensional structure where each element in the sequence is associated with a weight. We transform the DNA microarray data, as well as pattern-based queries, into weighted-sequences, and use subsequence matching algorithms to retrieve from the database all genes that match the query pattern. We demonstrate, using both synthetic and real-world data sets, that our method is effective and efficient.

1 Introduction

With the advent of DNA microarray technology, gene expression in an organism can be monitored on a genome-wide scale, allowing the transcription levels of many genes to be measured simultaneously. It is widely believed that the availability of large gene expression databases will bring a huge impact on modern biology in the near future. As a result, we will be facing an explosion of gene expression data that may dwarf even the human sequencing projects [1, 4].

Analysis of large-scale microarray data is becoming one of the major bottlenecks in the utilization of the technology. 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 coherently under a subset of conditions, that is, they exhibit fluctuation of a similar shape when conditions change.

Much research has been done in mining and analyzing subspace patterns embedded in DNA microarrays. Most of the work, however, has centered on subspace pattern clustering, which endeavors to discover *all* possible pattern correlations in a DNA microarray. One notable limitation of subspace clustering algorithms is their scalability, which makes them inappropriate for large scale data analysis. Recently, several approaches that employ greedy algorithms to mine subspace patterns have been proposed, including the bicluster model [5] and the δ -cluster model [21]. Still, their complexity is beyond $O(n^2m)$ in the best case, where n is the number of rows and m is the number of columns of the DNA array.

With the increasing availability of large DNA microarray databases, efficient processing of queries based on gene expression patterns is becoming one of the important issues in utilizing the DNA microarray technique. In this paper, we identify two types of frequently asked queries against DNA microarray databases. Both types are based on *specific* patterns of gene expression levels.

We use a sample dataset to demonstrate the queries of interest. Table 1 shows a small portion of the Yeast expression data [17], where each entry d_{ij} – the expression level of gene i in sample j – is derived by transformation $d_{ij} = 100 \log \left(10^5 \frac{RedIntensity_{ij}}{GreenIntensity} \right)$ that scales the value into a range of 0–600.

Instead of mining all the patterns in the dataset, a typical search in the DNA microarray database tries to find all genes that conform to a specific pattern. For instance:

| | CH1I | CH1B | CH1D | CH2I | CH2B |
|-------|------|------|------|------|------|
| VPS8 | | | | | |
| SSA1 | | | | | |
| SP07 | | | | | |
| EFB1 | | | | | |
| MDM10 | | | | | |
| CYS3 | | | | | |
| DEP1 | | | | | |
| NTG1 | | | | | |

Figure 1. Expression data of Yeast genes, derived by transformation $d_{ij} = 100 \log \left(10^5 \frac{RedIntensity_{ij}}{GreenIntensity} \right)$

Example 1. Searching Patterns

Find all genes whose expression level in sample CH1I is around 315 ± 5 , in sample CH1B is around 280 ± 10 , in sample CH1D is around 30 ± 7 , and in sample CH2B is around 210 ± 5 .

The query can be expressed in SQL:

```
SELECT *
FROM dna-array
WHERE 310 <= CH1I <= 320
      AND 270 <= CH1B <= 290
      AND 23 <= CH1D <= 37
      AND 205 <= CH2B <= 215
```

The query shown in Example 1 searches the microarray database for a fixed pattern. We are more interested, however, in identifying a set of genes that respond to certain experimental perturbances in the same manner, although their responses may not be at the same level. For instance, as shown in Figure 2, the expression levels of three genes, VPS8, CYS3, and EFB1, rise and fall coherently under three different conditions, but their values of the three columns are not necessarily close. In other words, a more challenging type of query is the following:

Given a pattern defined in a subset of columns, what are the genes in the microarray databases whose values on these columns exhibit a strong correlation with the given pattern?

As an example, a query of interest might be in the following form:

Example 2. Searching Pattern Correlations

Find all genes whose expression level in sample CH1I is about 100 ± 5 units higher than that in CH2B, 280 ± 10 units higher than that in CH1D, and 75 ± 7 units higher than that in CH2I.

The query can also be expressed in SQL:

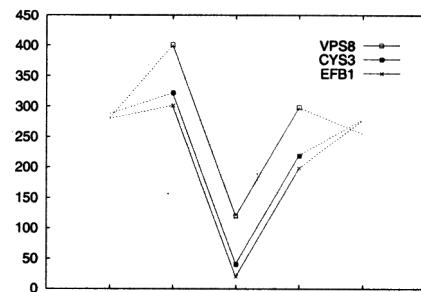


Figure 2. Expression levels of genes VPS8, CYS3, EFB1 rise and fall coherently in samples CH1I, CH1D, CH2B

```
SELECT *
FROM dna-array
WHERE 95 <= |CH1I - CH2B| <= 105
      AND 270 <= |CH1I - CH1D| <= 290
      AND 68 <= |CH1I - CH2I| <= 82
```

To answer both types of the queries, a brute force solution is to make a linear scan of the entire database. Needless to say, this approach is inefficient, considering the ever-increasing volume of the microarray data. The SQL query in Example 1 can be answered in sublinear time by creating an R+Tree index on the dataset. However, the real challenge lies in answering the SQL query in Example 2 efficiently. State-of-the-art database indexing techniques do not have a solution for this type of query. For instance, the R+Tree technique can not apply because R+Tree indexes on exact values, not on correlation of values. The difficulty is compounded by the fact that a DNA microarray can have hundreds of columns, and the WHERE clause in such queries can contain any subset of them. Indexing on each and every single column or a limited number of combinations of them can usually do no better than scanning the entire data set.

OUR CONTRIBUTIONS.

- We present a method to reduce the pattern correlation problem to the problem of weighted-subsequence matching [19]. This is achieved by transforming a relational table of numerical columns to a set of weighted sequences, and transforming a pattern correlation query (Example 2) to a weighted subsequence.
- We prove and demonstrate that the pattern correlation query can be answered in sublinear time

using the index structure designed for weighted-subsequence matching.

PAPER ORGANIZATION.

The rest of the paper is organized as follows. We introduce the weighted-sequence model in Section 2. Section 3 reviews a compact index structure and an algorithm framework designed for subsequence matching. In Section 4, we show that queries over DNA microarray databases can be expressed through weighted-subsequence matching. Section 5 shows that queries such as Example 2 can be answered in sublinear time. Experiments and results are reported in Section 6. We review related work in Section 7 and conclude in Section 8.

2 Weighted-Sequences

In this section, we introduce the weighted-sequence model. As we shall see in Section 4, the pattern correlation problem of gene expression levels can be solved under this model.

Definition 1. Weighted-Sequence

A weighted-sequence is a sequence of (symbol, weight) pairs.

$$\mathcal{T} = \langle (a_1, w_1), (a_2, w_2), \dots, (a_n, w_n) \rangle$$

Here, each a_i is a symbol, and $w_i \in \mathbb{R}$.

In this paper, we focus on sequences where weights are in ascending order, i.e., $w_i \leq w_j$ for $i < j$. We shall see later that DNA microarrays can be reduced to sequences in the ordered form. For the rest of the paper, we use weighted-sequence to denote a sequence whose items are associated with ascending weights.

We introduce some notations based on the above definition.

| | |
|------------------------------------|---|
| \mathcal{T} | a weighted-sequence |
| \mathcal{T}_i | the i -th item in sequence \mathcal{T} |
| $s(\mathcal{T}_i)$ | symbol of the i -th item |
| $w(\mathcal{T}_i)$ | weight of the i -th item |
| $ \mathcal{T} $ | length (number of items) of \mathcal{T} |
| $\ \mathcal{T}\ $ | weight range of \mathcal{T} , $\ \mathcal{T}\ = w(\mathcal{T}_{ \mathcal{T} }) - w(\mathcal{T}_1)$ |
| $\mathcal{T}' \subset \mathcal{T}$ | \mathcal{T}' is a (non-contiguous) subsequence of \mathcal{T} |
| ξ | window size |

Note that $\|\mathcal{T}\|$ is the weight difference between the first and last elements of \mathcal{T} . A subsequence of \mathcal{T} , possibly non-contiguous, is derived from \mathcal{T} by discarding

some of its items. We use $\mathcal{T}' \subset \mathcal{T}$ to indicate that \mathcal{T}' is a (non-contiguous) subsequence of \mathcal{T} .

A query sequence $\mathcal{Q} = \langle (b_1, 0), \dots, (b_m, w_m) \rangle$ is a special weighted-sequence in that the weight of the first item in the sequence, $w(\mathcal{Q}_1)$, is 0.

Definition 2. Weighted-Sequence Matching

A query sequence \mathcal{Q} matches sequence \mathcal{T} if there exists a (non-contiguous) subsequence $\mathcal{T}' \subset \mathcal{T}$ such that $|\mathcal{Q}| = |\mathcal{T}'|$, $s(\mathcal{Q}_i) = s(\mathcal{T}'_i)$, and $w(\mathcal{Q}_i) = w(\mathcal{T}'_i) - w(\mathcal{T}'_1)$, $\forall i \in 1, \dots, |\mathcal{Q}|$.

An example of weighted-sequences matching is shown in Figure 3, where a query sequence $\langle (a, 0), (b, 6), (c, 9) \rangle$ matches a weighted-subsequence of \mathcal{T} .

Apparently, sequence matching in Definition 2 requires that the weight difference between any two items in the matched subsequence is exactly the same as that of the corresponding items in the query sequence. This restriction can be relaxed to allow approximate matching.

Definition 3. Approximate Matching of Weighted-Sequences

Given a sequence \mathcal{T} , a query sequence \mathcal{Q} , and error $e_i \geq 0$, $i \in 1, \dots, |\mathcal{Q}|$, we say \mathcal{Q} approximately matches \mathcal{T} if there exists a (non-contiguous) subsequence $\mathcal{T}' \subset \mathcal{T}$ such that $|\mathcal{Q}| = |\mathcal{T}'|$, $s(\mathcal{Q}_i) = s(\mathcal{T}'_i)$, and $|w(\mathcal{Q}_i) - (w(\mathcal{T}'_i) - w(\mathcal{T}'_1))| \leq e_i$, $\forall i \in 1, \dots, |\mathcal{Q}|$.

3 Weighted-Sequence Matching

In this section, we review the weighted-sequence matching algorithm introduced by Wang et al. [19].

3.1 Overview

The index structure for finding (non-contiguous) subsequences that match the query sequence is called the *Iso-Depth Index*. The index construction algorithm is parameterized by a window size, ξ . The iso-depth structure embodies a compact index to all subsequences with a weight range less than ξ in a data sequence. Consequently, queries longer than ξ are broken down into multiple sub-queries and their results are joined to produce a final answer.

During index construction, a suffix tree is employed as an intermediary structure to facilitate the building of the iso-depth index. The suffix tree supports efficient matching of contiguous subsequences. However, if a query sequence is non-contiguous, for instance, $abc****def$, where $*$ stands for 'don't care', we need to search the subtree under c for up to 5 levels to find

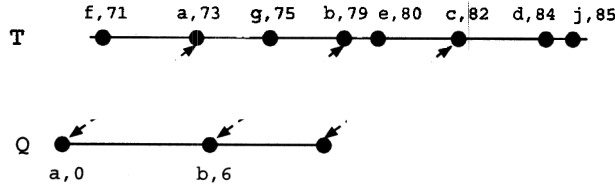


Figure 3. Query $Q = \langle (a,0), (b,6), (c,9) \rangle$ matches a non-contiguous subsequence of T , $\langle (a,73), (b,79), (c,82) \rangle$

all occurrences of d there. The iso-depth structure, however, enables us to *jump* to such d 's right away without searching the subtree.

3.2 The Index Structure

Given a data sequence D , we place a moving window of size ξ on D . For each moving window, we apply function f defined in (1) on the sequence inside the moving window to encode it into a one dimensional sequence. Let $\mathcal{T} = \langle \mathcal{T}_1, \dots, \mathcal{T}_n \rangle$ be the sequence inside a moving window ($|\mathcal{T}| < \xi$). We have $f(\mathcal{T}) = \mathcal{S}$, where \mathcal{S} is a one dimensional sequence.

$$f(\langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle) = \langle \mathcal{S}_1, \dots, \mathcal{S}_n \rangle \quad (1)$$

where

$$\mathcal{S}_i = \begin{cases} s(\mathcal{T}_i)_0 & : i = 1 \\ s(\mathcal{T}_i)_{w(\mathcal{T}_i) - w(\mathcal{T}_{i-1})} & : i > 1 \end{cases}$$

and $s(\mathcal{T}_i)$ is the symbol of the i^{th} item of \mathcal{T} .

Let \mathcal{T} be the sequence in the k^{th} moving window on D . We insert $f(\mathcal{T}) = \mathcal{S}$ into a suffix tree. Assuming the insertion ends up at node v , we append value k to list L_v , the offset list of node v . After all encoded sequences are inserted, we make a depth-first traversal of the tree. We assign sequential IDs (starting with 0, which is assigned to the root) to the tree nodes in the depth-first traversal order. In addition, we also record for each node the largest ID of its descendents. More specifically, each node v is assigned a pair of numbers, (v_s, v_m) , where v_s is the ID of node v , and v_m is the ID of the right-most descendent of v . Based on the numbering, the ID of any descendent of v is between v_s and v_m .

For a given node v , let $v_{\mathcal{P}}$ be the path descending from the root node to node v . We use $\|v_{\mathcal{P}}\|$, or simply $\|v\|$, to denote the distance between the root and v . $\|v\|$ can be derived by simply summing up the subscripts of symbols in sequence $v_{\mathcal{P}}$.

Figure 4 shows a suffix tree and several horizontal links, or the iso-depth links. An iso-depth link is created for each (x, d) pair, where x is a symbol, and

$d = 1, \dots, \xi$. As we visit the nodes in depth-first order, we append each node to an iso-depth link. Assuming the arc that points to node v in the suffix tree is labeled with x_k , we append v to the iso-depth list for pair $(x, \|v\|)$. Thus, an iso-depth link is composed of nodes that have the same distance from the root. The iso-depth links have the following property.

Property 1. The Iso-depth Property

1. Nodes in an iso-depth link are sorted by their IDs in ascending order;
2. A node's descendents that appear in an iso-depth link are contiguous in that link. More formally, let $v \dots w \dots u$ be three nodes in an iso-depth link, in that order. If r is an ancestor of both v and u , then r is also an ancestor of w .

Proof.

1. Nodes are appended during the depth-first traversal when node IDs of increasing values are generated.
2. Since r is an ancestor of both v and u , we have $r_s < v_s \leq r_m$ and $r_s < u_s \leq r_m$. From $v_s < w_s < u_s$, we get $r_s < w_s < r_m$, which means r is an ancestor of w .

□

In Figure 4, each node v , represented by pair (v_s, v_m) , appears in only one iso-depth link. Each iso-depth link is organized as an array of (v_s, v_m) pairs. We store those arrays, as well as the offset list for each node, in the index file. Note that we do not store the tree structure (parent-child links) in the index; we show below that iso-depth links alone contain complete information for efficient subsequence matching.

Algorithm 1 summarizes the index construction procedure discussed above.

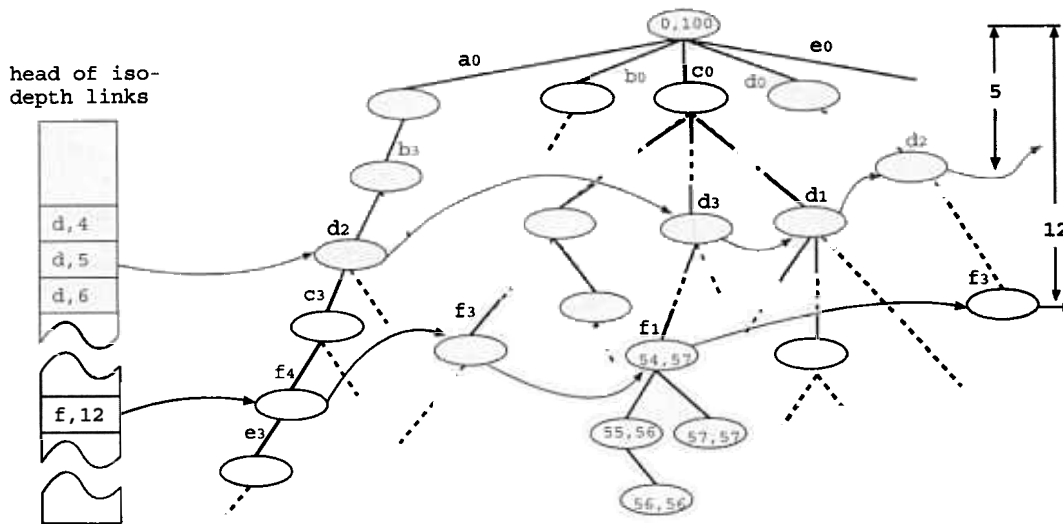


Figure 4. Suffix trie with iso-depth links for each symbol. Each node v is represented by pair (v_s, v_m) , where v_s is the ID of v , v_m is the largest ID under v .

3.3 Subsequence Matching

Algorithm 2 presents the outline of searching a given weighted subsequence in an index file.

For instance, we have a query sequence Q

$$Q = \langle (c, 0), (d, 5), (f, 12) \rangle$$

Suppose we have located all the nodes representing a prefix of Q , $\langle (c, 0), (d, 5) \rangle$. Let v be one of such nodes. In order to reach from v the next symbol d , we consult the iso-depth link for $(f, 12)$, which contains all the nodes of f that are 12 units away from the root. However, we are only interested in those that are descendants of v . The nodes in the iso-depth link are sorted by their IDs in ascending order. According to the 2nd iso-depth property, those descendants are contiguous in the link. Thus, we only need to find the first node whose ID is larger than v_s , the ID of v .

After we successfully match all the symbols in Q , we reach a set of nodes that correspond to the last symbol in Q . Let v be one of such nodes. The offset lists of node v and all nodes under v contain positions where Q occurs in \mathcal{D} . Thus, we return offsets $L_{v_s}, L_{v_s+1}, \dots, L_{v_m}$, which are stored sequentially in the index file.

Figure 5 illustrates the process of searching for subsequence Q using the disk-resident iso-depth index. It first consults iso-depth links, then it returns offsets in the offset lists. It shows that iso-depth links contain complete information for subsequence matching.

4 Mapping DNA Microarrays to Weighted-Sequences

In this section, we show how DNA microarrays can be converted to weighted sequences and how pattern correlation queries such as the one in Example 2 can be answered through subsequence matching.

We transform each record in a numerical relation into a structured sequence. For instance, the first record (gene VPS8) in the Yeast DNA microarray (Figure 1) can be represented by a list of column-value pairs,¹ which is shown in Table 1. We omit NULL values in the transformation.

We sort the column-value pairs in the list by the values in ascending order to derive a sequence with ascending weights, which has column names as symbols, and expression levels as weights. Concatenating all the weighted-sequences thus derived from each record, and delimiting them by a special item, for instance, (NULL, 0), we have transformed the entire table into a long sequence.

We call the resulting long sequence (shown in Table 1) a *generalized* weighted-sequence, as weights of its items are only sorted locally between the (NULL, 0) delimiters.

The problem of querying relational tables with numerical columns is now equivalent to (approximate) weighted-subsequence matching. For instance, the

¹For presentation simplicity, we use only 5 columns of the Yeast gene array. A widely available Yeast micro-array [17], which is also used in our experiments, has 17 columns.

Input: \mathcal{D} : weighted-sequence, ξ : window size

Output: F : index of \mathcal{D}

for all sequences \mathcal{T} in moving window of size ξ do

 | insert $f(\mathcal{T})$ into a suffix trie;

make a depth-first traversal of the tree;

for each node v encountered in the traversal do

 | label node v by (v_s, v_m) ;

 | let a_k be the tree arc that points to v ;

 | append (v_s, v_m) to iso-depth list $(a, \|v\|)$;

index file F contains two parts:

▷ iso-depth links, where node v is represented by a pair (v_s, v_m) ;

▷ offset list $L[0..m]$ for node $0..m$, where m is the largest node ID.

Algorithm 1: Index Construction

Input: \mathcal{Q} : query sequence; $e_1, \dots, e_{|\mathcal{Q}|}$: tolerance; F : index file for \mathcal{D}

Output: offsets in \mathcal{D} where \mathcal{Q} occur

Let $\mathcal{Q} = \langle (q_1, 0), \dots, (q_i, w_i), \dots \rangle$;

$v \leftarrow$ root's child node under arc q_1 ;

$search(v, 1)$;

Function $search(v, i)$

if $i < |\mathcal{Q}|$ then

 | $i \leftarrow i + 1$;

 | **for each iso-depth link I in $(q_i, w_i \pm e_i)$ do**

 | /* Perform binary search in I to locate the 1st node r such that $r_s \geq v_s$ */

 | **for each node $r \in I$ whose ID $\in [v_s, v_m]$ do**

 | $search(r, i)$;

 | **end**

 | **end**

else

 | output $L[v_s..v_m]$, offset lists of node v and all nodes under v ;

end

Algorithm 2: Subsequence Matching

DNA micro-array query given in Example 2 can be paraphrased into the following after the array is transformed into a sequence:

Example 3. Querying DNA Micro-array by Sequence Matching

Let \mathcal{T} be the generalized weighted-sequence converted from the Yeast DNA micro-array. Find all subsequences of \mathcal{T} that match query

$\mathcal{Q} = \langle (\text{CH1D}, 0), (\text{CH2B}, 180), (\text{2H2I}, 205), (\text{CH1I}, 280) \rangle$

with tolerance $e_1 = 0$, $e_2 = 5$, $e_3 = 7$, and $e_4 = 10$.

To support pattern correlation queries against DNA microarray datasets using the weighted-sequence

matching algorithm, we need to take care of several issues:

- **Discretization.** Gene expression levels are usually represented by real numbers. We discretize them into equi-width units. Let $\mathcal{S} = f(\mathcal{T})$ be a one-dimensional sequence inserted into the suffix tree. After discretization, the subscripts of each element in \mathcal{S} shall have a value range of $1, \dots, \xi$, where ξ is the window size. The queries, as well as the tolerances associated with the queries, are discretized in the same way.
- **Moving window size.** For DNA microarray data

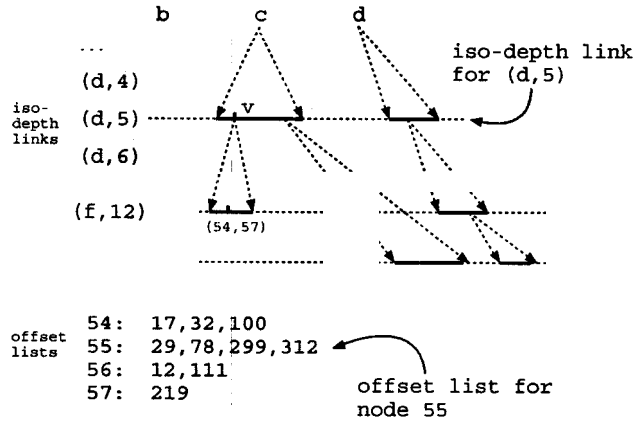


Figure 5. Searching for weighted subsequence $Q = \langle (c, 0), (a, 6), (b, 12) \rangle$ using iso-depth index.

Gene VPS8: (CH1I, 401), (CH1B, 281), (CH1D, 120), (CH2I, 275), (CH2B, 298)
 ↓
 Sort by value: $\langle (CH1D, 120), (CH2I, 275), (CH1B, 281), (CH2B, 298), (CH1I, 401) \rangle$
 ↓
 Concatenate all records: $\underbrace{(CH1D, 120), \dots, (CH1I, 401)}_{1st\ record}, (NULL, 0), \underbrace{(CH1D, 109), \dots, (CH2I, 580)}_{2nd\ record}, (NULL, 0)$

Table 1. Reducing Numerical Tables to Weighted-Sequences

sets, we set the window size no larger than the value range of the expression levels. Such a window size will include every column of a single gene in one window. We can choose, for instance, $\xi = 600$ for the Yeast DNA microarray in Table 1, since the data are in the range of 0–600. If we choose a ξ less than the value range, then there is a possibility that we need to break down a query into multiple sub-queries.

- The (NULL, 0) boundary. One thing special about the sequences converted from microarrays is that genes are separated by the (NULL, 0) delimiters in the sequence. When we index subsequences, there is no need for the moving window to cross the (NULL, 0) boundary.

In the rest of this section, we use an example to demonstrate how pattern correlation queries are implemented on top of the weighted-sequence matching algorithm.

Example 4. Let a sequence database \mathcal{D} be composed of the following symbol/weight pairs (discretized). Let

$\xi = 15$ be the window size.

$\mathcal{D} = \langle (a, 6), (b, 9), (d, 11), (c, 14), (f, 18), (e, 21), (NULL, 0), (c, 25), (d, 32), (b, 33), (a, 34), (e, 37), (f, 52), (NULL, 0) \rangle$

Since queries are constrained by the windows size, we only need to index subsequences $\mathcal{T} \subset \mathcal{D}$ where $\|\mathcal{T}\| \leq \xi$. We create a moving window of size ξ over \mathcal{D} . As we move the window along \mathcal{D} , we find the following subsequences in the window (windows do not cross NULL boundaries):

- | | |
|-----|--|
| 1: | (a, 6), (b, 9), (d, 11), (c, 14), (f, 18), (e, 21) |
| 2: | (b, 9), (d, 11), (c, 14), (f, 18), (e, 21) |
| 3: | (d, 11), (c, 14), (f, 18), (e, 21) |
| 4: | (c, 14), (f, 18), (e, 21) |
| 5: | (f, 18), (e, 21) |
| 6: | (c, 25), (d, 32), (b, 33), (a, 34), (e, 37) |
| 7: | (d, 32), (b, 33), (a, 34), (e, 37) |
| 8: | (b, 33), (a, 34), (e, 37) |
| 9: | (a, 34), (e, 37) |
| 10: | (e, 37), (f, 52) |
| 11: | ... |

| | |
|-----|--------------------------------|
| 1: | $a_0, b_3, d_2, c_3, f_4, e_3$ |
| 2: | b_0, d_2, c_3, f_4, e_3 |
| 3: | d_0, c_3, f_4, e_3 |
| 4: | c_0, f_4, e_3 |
| 5: | f_0, e_3 |
| 6: | c_0, d_7, b_1, a_3, e_3 |
| 7: | d_0, b_1, a_3, e_3 |
| 8: | b_0, a_3, e_3 |
| 9: | a_3, e_3 |
| 10: | e_0, f_{15} |
| 11: | ... |

Figure 6. Encoded weighted-sequences with an expanded symbol set. Transformed from sequence in each window.

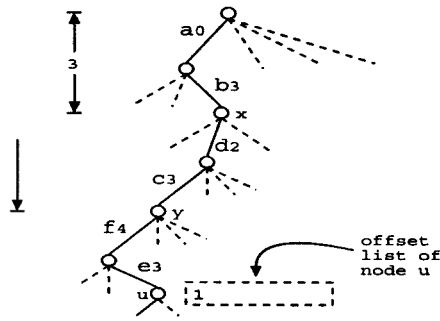


Figure 7. Insertion of $f(\mathcal{T}) = \langle a_0, b_3, d_2, c_3, f_4, e_3 \rangle$ ends up at node u . The offset of \mathcal{T} in the original sequence \mathcal{D} is appended to the offset list of u .

The resulting one-dimensional sequences are shown in Table 6.

As we can see, the subscripts of the symbols in the one-dimensional sequence \mathcal{S} represent intervals (weight differences) between two adjacent symbols in the original weighted sequence. Let $f(\mathcal{T}) = \mathcal{S}$. The weight range of \mathcal{T} is the sum the subscripts in \mathcal{S} , i.e., $\|\mathcal{T}\| = \sum_{v_i \in \mathcal{S}} i$.

The encoded sequences have an expanded symbol set. Take the 1st and the 2nd sequence in Figure 6 for example. Symbol b_3 in the 1st sequence and symbol b_0 in the 2nd sequence are two different, independent symbols. We insert each sequence into a suffix tree by following the arcs in the sequence. The insertion of the 1st sequence $f(\mathcal{T}) = \langle a_0, b_3, d_2, c_3, f_4, e_3 \rangle$ is shown in Figure 7. Each node in the tree has an *offset list*. Assuming the insertion of $f(\mathcal{T})$ leads to node u , which is pointed to by arc e_3 , we append to the offset list of

u the position of \mathcal{T} in the original sequence \mathcal{D} , which in this case, is 1, since \mathcal{T} appears in the 1st window of \mathcal{D} .

5 Complexity Analysis

Let \mathcal{D} be a data sequence, \mathcal{Q} a query sequence. We use S_j to denote the number of nodes on the j -th level of the suffix tree, $j = 0, \dots, \xi$. Apparently, we have $S_0 = 1$, which is the root node. Also, we have $S_1 = |\mathcal{A}|$, where \mathcal{A} is the set of symbols that appear in the sequence data. Sequence matching starts with nodes on level one, which are made memory resident, since there are only $|\mathcal{A}|$ of such nodes.

Assuming query sequence \mathcal{Q} has k elements, we estimate the number of nodes we need to visit on different levels of the suffix tree in order to answer query \mathcal{Q} .

$$\mathcal{Q} = \langle (a_1, 0), (a_2, t_2), (a_3, t_3), \dots, (a_k, t_k) \rangle$$

After we visit node a_1 on the first level, we consult iso-depth link (a_2, t_2) . Let's assume symbols have the same frequency and they are uniformly distributed in \mathcal{D} ; the total number of nodes on (a_2, t_2) is approximately $S_{t_2}/|\mathcal{A}|$. Since we are only interested in those nodes under node a_1 on the first level, we only need to visit an average of $S_{t_2}/|\mathcal{A}|^2$ nodes on level t_2 .

For each of the $S_{t_2}/|\mathcal{A}|^2$ nodes on level t_2 , we visit its descendents on link (a_3, t_3) . On average, each node on level t_2 has less than S_{t_3}/S_{t_2} descendents on level t_3 , and only $1/|\mathcal{A}|$ of them are associated with symbol a_3 . Thus, the total number of nodes we need to visit on level t_3 is:

$$\frac{S_{t_2} S_{t_3}}{|\mathcal{A}|^2 S_{t_2}} \frac{1}{|\mathcal{A}|} = \frac{S_{t_3}}{|\mathcal{A}|^3}$$

Consequently, the number of nodes we need to visit in order to answer query \mathcal{Q} is:

$$N = 1 + \frac{S_{t_2}}{|\mathcal{A}|^2} + \dots + \frac{S_{t_i}}{|\mathcal{A}|^i} + \dots + \frac{S_{t_{k-1}}}{|\mathcal{A}|^{k-1}} + a \quad (2)$$

where a is the number of nodes that contain answers of the query.

After n insertions ($|\mathcal{D}| = n$) to the suffix tree, the expected number of nodes on the j -th level, S_j , comes to:

$$S_j = \frac{1}{p_j} (1 - (1 - p_j)^n) \quad (3)$$

where p_j is the probability that any two weighted-sequences with j elements are the same. To estimate p_j , we consider the simplest case where there are no "gaps" in the data sequence, i.e., $w(D_i) - w(D_{i-1}) = 1$.

Clearly, one has $p_j = 1/|A|^j$. However, if there are “empty slots” in \mathcal{D} , we have $p'_j = ((1 - \delta)^2 + \frac{1}{|A|}\delta^2)^j$, where density $\delta = \frac{|\mathcal{D}|}{\|\mathcal{D}\|} \leq 1$. As long as $\delta \leq 1 - \frac{2}{|A|+1}$, which is easy to guarantee for any realistic value of $|A|$, we have $p'_j > p_j = 1/|A|^j$.

According to Formula 3, we have $S_j \leq 1/p_j = |A|^j$. However, for a fixed data sequence \mathcal{D} , as j increases, $p_j \rightarrow 0$, and $n \ll 1/p_j$. Thus, in the lower levels of the suffix tree, we have:

$$\lim_{p_j \rightarrow 0} S_j = \lim_{p_j \rightarrow 0} n(1 - p_j)^{n-1} = n$$

Assuming $|A|^{t_i} \leq n \leq |A|^{t_{i+1}}$, from Formula 2, we get:

$$\begin{aligned} N &\leq 1 + \frac{|A|^{t_2}}{|A|^{t_2}} + \frac{|A|^{t_i}}{|A|^{t_i}} + \frac{n}{|A|^{t_{i+1}}} + \frac{n}{|A|^{t_k}} + a \\ &\leq \sum_{j=2}^i |A|^{t_j-j} + \frac{n}{|A|^i(|A|-1)} + a + 1 \end{aligned}$$

In summary, assuming we are given a sequence dataset \mathcal{D} with n elements ($|\mathcal{D}| = n$), a query sequence \mathcal{Q} with k elements ($|\mathcal{Q}| = k$), and window size ξ . If $n > |A|^\xi$, we are able to find all matchings in \mathcal{Q} with at most $\frac{1}{B} \sum_{j=2}^{k-1} |A|^{t_j-j} + a$ disk accesses on average, where B is the page size. If $n \leq |A|^\xi$, then the number of disk accesses is linear in terms of n , but is reduced by factor of about $1/|A|^i$ from a linear scan, where exponent i depends on the query sequence ($i \geq 2$).

6 Experiments

We experimented our index structure on both synthetic and real life data sets. The algorithm is implemented on a Linux machine with a 700 MHz CPU and 256 MB main memory.

6.1 Data Sets

Each element in a synthetic data sequence is represented by a pair of integers (symbol, weight), thus, the size of a dataset comes to $8n$ bytes, where n is the number of elements it has. In our experiments, we use disk page size $B = 2K$ bytes.

GENE EXPRESSION DATA Gene expression data are being generated by DNA chips and other micro-array techniques. The data set is presented as a matrix. Each row corresponds to a gene and each column represents a condition under which the gene is developed. Each entry represents the relative abundance of the mRNA

of a gene under a specific condition. The yeast micro-array [17] 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. Accordingly, we use a moving window of size $\xi = 40$, so that all 17 expression levels of a gene fit into one window. The mouse cDNA array [6] is 535,766 in size (10,934 genes under 49 conditions) and it is pre-processed in the same way.

SYNTHETIC DATA We generate synthetic data to simulate queries on large datasets. In addition to symbol size $|A|$, sequence length $|\mathcal{D}|$, the synthetic data generator also simulates the distribution of symbols and weights in the sequence. We generate nondecreasing weights from zero in such a way that the number of elements in a unit window follows either uniform or Poisson distribution. A sample dataset named D100K-A40-P10, for instance, indicates that the synthetic sequence has 100K elements, 40 different symbols, and the weight difference of two adjacent elements in the sequence follows a Poisson distribution with parameter $\lambda = 10$.

6.2 Performance Analysis

We start with experiments on scalability. The performance curves presented in Figure 8 show that iso-depth index scales much better than two alternative algorithms. The comparisons are carried out on synthetic datasets, D?-A200-U10, which range in size from 200 thousand to 10 million elements. All 200 symbols in the synthetic sequences have the same occurrence rate, and the average weight difference between two adjacent symbols is 10.

We ask fixed-format queries (3 evenly separated elements in a window of size $\xi = 45$) against the datasets. The Y axis of Figure 8 represents the number of pages accessed in log scale. The alternative algorithms used in the comparisons is brute-force linear scan, which is often used to answer queries such as Example 2. The iso-depth index is orders of magnitude faster.

Next, in Figure 9, we study the impact of query length on the performance, using yeast and mouse DNA micro-array data. We generate random queries with 2 to 5 evenly separated events within the span of a moving window ($\xi = 40$). Figure 9 shows the average number of node accesses and disk accesses. The results indicate that there our approach is robust as the query size becomes larger. This is the result of increased selectivity of longer queries. In Figure 10, we repeat the experiment on synthetic dataset D5000K-A100-P10,

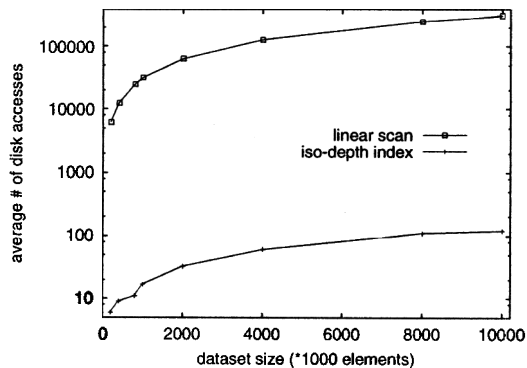


Figure 8. Overall comparisons with alternative algorithms over queries on synthetic datasets D?-A200-U10 varying sequence length.

whose weights follow a Poisson distribution with parameter $\lambda = 20$. We used moving windows of size $\xi = 50$, and the results are similar.

We also study the impact of different query forms on the performance. We generate queries in the form $\langle (a_0, 0), (a_1, t), (a_2, \xi) \rangle$ with a varying t , where a_0 , a_1 , and a_2 are random symbols in dataset D1000K-A100-U10, and $\xi = 60$ is the window size. Figure 11 presents the results. When t becomes larger, more disk accesses occur as the number of a_1 nodes under a_0 increases dramatically.

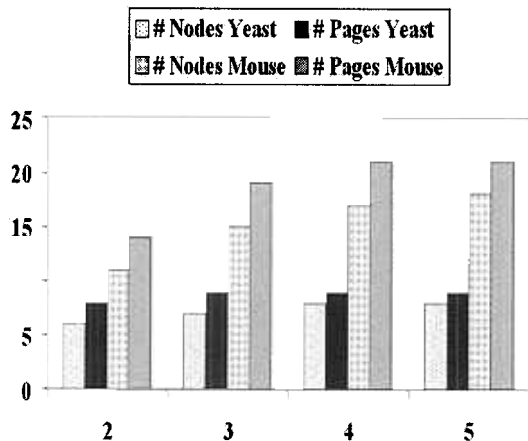


Figure 9. Random queries against DNA micro-arrays of Yeast and Mouse gene expression, with varying number of elements in the query from 2 to 5.

7 Related Work

There has been much research on indexing substrings. 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 [10] also provide fast searches on text databases. String B-tree [8, 9] uses an index structure similar to that of B-tree to overcome the unbalanced tree topology problem of suffix trees.

The above index structures, however, are not adequate to solve the problems mentioned in the previous section, because they only provide fast accesses for searching contiguous subsequences in a string database. More specifically, the relative positions of two elements in a string is also used to embody the distance between them, while in a weighted-sequence, the distance between two elements is expressed explicitly in another dimension, the weights.

Similarity based subsequence matching [7, 15] has been a research focus for applications such as time series databases. The basic idea is to map each data sequence into a small set of multidimensional rectangles in feature space. Traditional spatial access methods [11, 3] are then used to index and retrieve these rectangles. Here, retrieval is based on similarity of the time-series within a continuous time interval. The method can not be applied to solve the weighted-sequence problem since the pattern to retrieve is usually a non-contiguous subsequence in the original sequence.

Recently, the problem of *exact* matching for multidimensional strings is proposed in [12]. Strings are mapped to real numbers based on their lexical order. Later, these multidimensional points are indexed using R-trees [11]. This technique works efficiently for queries such as “find a person whose name begins with Sri and telephone number begins with 973”. Our concern in this paper, however, is to index objects that match a given pattern instead of exact values.

To the best of our knowledge, there has been little research in fast retrieval of numerical patterns in relational tables. The above mentioned techniques can not be applied directly to solve this problem, largely because they only handle one-dimensional series. On the other hand, much research has been devoted to find *frequent* patterns in large databases [16, 2]. These methods typically scan a data set multiple times in order to find patterns whose occurrence level is beyond a threshold. Recent advances in biology, such as the DNA micro-array technique, also fuel this need. As a result, several clustering-based algorithms [5, 20] have been introduced to attack this problem.

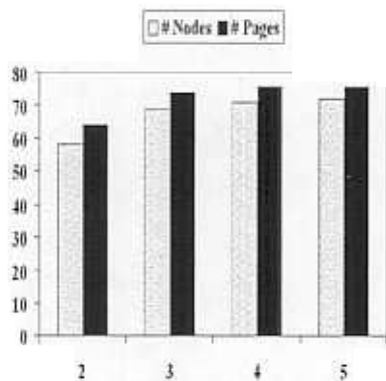


Figure 10. Random queries on dataset D5000K-A100-P10, with query size from 2 to 5.

8 Conclusion and Future Work

With the increasing availability of large DNA microarray databases, efficient processing of queries based on gene expression patterns is becoming one of the important issues in utilizing the DNA microarray technique. In this paper, we identified two types of pattern-based queries that are frequently posed against DNA microarray databases. In particular, the second type of query, which is of more interest in DNA microarray analysis, is not supported well by current database indexing techniques.

We showed in our paper that searching pattern correlation among a subset of columns is equivalent to finding subsequence matches among a set of weighted-subsequences. This is achieved by transforming DNA microarray datasets to weighted sequences. Pattern correlation queries, as well as the error tolerances associated with the queries, are transformed to weighted sequences in the same way. Thus, pattern correlation queries can be implemented on the index structure called iso-depth index designed for fast retrieval of weighted-subsequences in large datasets. Experimental results show that the index structure achieves orders of magnitude speedup over linear scan.

Queries based on pattern correlations instead of on exact values lead to a lot of potential research topics. One of our current work focuses on nearest neighbor (NN) search using pattern correlation as the distance metric. The difference between NN queries and the pattern correlation queries discussed in this paper is that NN queries do not specify a fixed subset of columns in the query. A gene satisfies the query criteria as long as its expression levels under an arbitrary subset of conditions are correlated with the expression levels of the query object under the same set of condi-

tions.

References

- [1] Alison Abbott. Bioinformatics institute plans public database for gene expression data. *Nature*, 398:646, 1999.
- [2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, 1995.
- [3] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.
- [4] Alvis Brazma, Alan Robinson, Graham Cameron, and Michael Ashburner. One-stop shop for microarray data. *Nature*, 403:699–700, 2000.
- [5] Y. Cheng and G. Church. Biclustering of expression data. In *Proc. of 8th International Conference on Intelligent System for Molecular Biology*, 2000.
- [6] 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 U.S.A*, 98, pages 2199–2204, 2001.
- [7] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

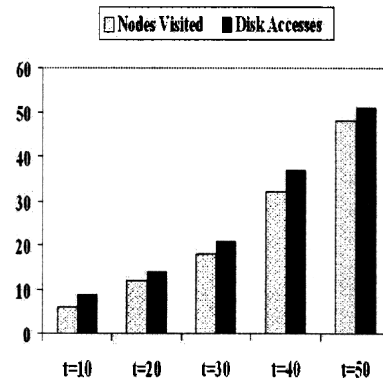


Figure 11. Query $\langle (a_0, 0), (a_1, t), (a_2, 60) \rangle$ on dataset D1000K-A100-U10, varying t from 10 to 50.

- [8] Paolo Ferragina and Roberto Grossi. Fast string searching in secondary storage: Theoretical developments and experimental results. In *Proceedings of the ACM SODA*, pages 373–382, Atlanta, 1996.
- [9] Paolo Ferragina and Roberto Grossi. The string B-tree: a new data structure for string search in external memory and its applications. *Journal of the ACM*, 46(2):236–280, 1999.
- [10] 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.
- [11] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.
- [12] H. V. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. In *SIGMOD*, pages 403–414, 2000.
- [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] 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.
- [16] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.
- [17] 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.
- [18] E. Ukkonen. Constructing suffix-trees on-line in linear time. *Algorithms, Software, Architecture: Information Processing*, pages 484–92, 1992.
- [19] H. Wang, C. Perng, W. Fan, S. Park, and P. Yu. Indexing weighted-sequences in large databases. Technical Report Research Report, IBM T. J. Watson Research Center, Hawthorne, NY, February 2002.
- [20] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD*, 2002.
- [21] 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.