

# The $S^2$ -Tree: An Index Structure for Subsequence Matching of Spatial Objects

Haixun Wang and Chang-Shing Perng

IBM T. J. Watson Research Center  
Yorktown Heights, NY 10598  
haixun@us.ibm.com perng@us.ibm.com

**Abstract.** We present the  $S^2$ -Tree, an indexing method for subsequence matching of spatial objects. The  $S^2$ -Tree locates subsequences within a collection of *spatial sequences*, i.e., sequences made up of spatial objects, such that the subsequences match a given query pattern within a specified tolerance. Our method is based on (i) the string-searching techniques that locate substrings within a string of symbols drawn from a discrete alphabet (e.g., ASCII characters) and (ii) the spatial access methods that index (unsequenced) spatial objects. Particularly, the  $S^2$ -Tree can be applied to solve problems such as subsequence matching of time-series data, where features of subsequences are often extracted and mapped into spatial objects. Moreover, it supports queries such as “what is the longest common pattern of the two time series?”, which previous subsequence matching algorithms find difficult to solve efficiently.

## 1 Introduction

The sequence of objects can endow it with some special significance that an unsequenced grouping of the same objects could never convey. In this paper, we focus on the design of fast searching methods that will search a database of sequences of text, spatial, or multimedia objects to locate those that match a query subsequence of objects. Such sequences can be 1-dimensional time series, digitized voice or music, video clips, trail of mobile objects and so on.

Specific applications that will benefit from our method include the following:

- Time series databases. The efficient matching [4] of time series data often relies on some distance-preserving transform, such as the Discrete Fourier transform (DFT), which extracts  $f$  features from sequences (e.g., the first  $f$  DFT coefficients), thus mapping them into points in the  $f$ -dimensional feature space.
- Content-based image querying. A similarity retrieval algorithm for image databases proposed in [20], for example, extracts image regions and uses Harr wavelet[22] to compute their signatures by mapping them to some multidimensional space.
- Content-based analysis, indexing, and retrieval of audio or video sequences. For instance, *Video-Trails'* approach to analyzing a video clip involves first generating a trail of points in a multi-dimensional space where each point is derived from physical features of a single frame in the video clip[24].
- Spatio-temporal databases, which deal with geometries changing over time[28].

The above databases and the queries processed against them have the following characteristics in common:

- Entities in the databases are either spatial objects, or can be converted into spatial objects through feature extraction. Examples of feature vectors are color histograms[6], Fourier vectors[7], text descriptors[26], etc. In some applications, the mapping process does not yield point objects, but extended spatial objects in high-dimensional space[27].
- There exists a (total or partial) order among the entities in the database. Objects in time series databases, audio, and video sequences are ordered by time. In content-based image querying, an image is often decomposed into a set of sub images which can be partially ordered by their relative positions in the original image. For the rest of the paper, we assume there is a total order among the entities in the database.

Taking advantage of the above characteristics, we find that the task of subsequence matching against databases of sequences of text, spatial, or multimedia objects is essentially the following problem: Given a query spatial sequence, search a set of spatial sequences to locate subsequences that are similar to the query sequence.

However, traditional database indexing techniques are inadequate for this purpose. There is currently much excellent work in indexing multidimensional data, including geometric hashing[25], grid-based index structures[15], and the R-tree family[9, 8] index structures. These spatial access methods, are designed to index *unsequenced* spatial objects. The order among the entities in the database is not taken into consideration when the index structures are created and hence no effective retrieval method in terms of subsequence matching of spatial objects is supported.

Our work extends the substring matching technique to spatial sequences. We propose a new index structure, the  $S^2$ -Tree, that can be applied to search databases of different contents when the features of the data are extracted into sequences of multidimensional objects. It also supports implementation of new SQL predicates, for example, *sound like* and *look like*, which are similar to the standard *like* predicate for substring matching, for queries in multimedia databases. In this paper, we focus on time series, where temporal patterns are usually mapped to feature vectors in some high-dimensional space and queries are processed against a database of those feature vectors[4].

The organization of the rest of the paper is as follows. In the next section we review some background material, including spatial access methods and the suffix tree. In Section 3 we propose our algorithm for fast subsequence matching of spatial objects. In Section 4, we use our algorithm to solve subsequence matching problem for time series databases. Section 5 contains experiments that show the effectiveness of our algorithms.

## 2 Background

**Spatial Access Methods** The R-tree can be viewed as an extension of the B-tree to multi-dimensions. The  $R^*$ -tree[8] improves the R-tree by introducing a policy called *forced reinsert*: when a node overflows, it is not split right away, but a portion of the entries are removed and reinserted into the tree. The  $R^*$ -tree also refines the node splitting policy of the R-tree by taking overlapping area and region perimeter into consideration. Another modification of the R-tree, called X-tree[5], is particularly well suited for indexing high-dimensional data. The main idea of the X-tree is to avoid overlap of bounding boxes in the directory by using a new organization of the directory which is optimized for high dimensional space. Instead of allowing splits that introduce high overlaps, X-tree postpones node splitting by introducing supernodes, i.e., nodes larger than the usual block size.

**Suffix Trees** A *suffix tree*[1] embodies a compact index to all the distinct, non-empty substrings of a given string. The overall space requirement of the suffix tree is linear in the length of the

string it represents. Various approaches of building the substring index in linear time have been developed. McCreight’s algorithm builds a suffix tree in linear time and is space efficient[1]. Ukkonen [2] developed a linear-time, on-line suffix tree construction algorithm.

Searching for all instances of a substring  $S$  in a suffix tree is easy since the symbols in  $S$  define a path down the suffix tree. Following this path, if we encounter a  $\$$  before reaching the end, then  $S$  is not in the tree. If we end up at a node  $x$  then  $S$  occurs at least once. Moreover the places where  $S$  can be found are given by the pointers in all the leaves in the subtree rooted at  $x$ .

### 3 The $S^2$ -Tree

The  $S^2$ -Tree<sup>1</sup> is motivated by the fact that searching substrings in a suffix tree takes an average  $O(a + \log n)$  disk accesses<sup>2</sup>, where  $a$  is the size of the answer set. It would be desirable if the same technique can be used to solve the problem of subsequence matching of spatial objects and reduce its complexity.

The major differences between spatial sequences and text strings are:

1. The alphabet of text strings usually consists of only a few discrete symbols (e.g. ASCII set); spatial sequences, however, do not have a pre-defined “alphabet”;
2. There is no relationship among symbols in a text string. On the other hand, quite a few relationships could exist between two spatial objects, for example, one spatial object *contains* or *overlaps* another spatial object.

The  $S^2$ -Tree bridges the gap between spatial sequences and text strings by creating an alphabet that encodes not only every spatial object but also its (containment) relationship with other spatial objects. The  $S^2$ -Tree is a combination of two tree structures: (i) The X-tree, which provides a clustering method of spatial objects. According to the clustering, objects are converted into binary encodings that embody the containment relationship. (ii) The suffix tree, which implements subsequence matching on the binary sequences.

**Notation** For the rest of the paper, we shall use the following notational conventions. Unless otherwise specified, we use uppercase letters,  $Q, R, S$ , to denote spatial sequences and we use lowercase letters,  $a, b, c$ , to denote *minimum bounding rectangles* (MBRs) of spatial objects.

- $|S|$  the length of spatial sequence  $S$ .
- $S[i]$  the  $i^{th}$  entry of spatial sequence  $S$ .  $S[i]$  is a spatial object represented by its MBR.
- $S[i, j]$  a subsequence that includes entries in position  $i$  through  $j$ .
- $a \subseteq b$  MBR  $b$  contains MBR  $a$ .
- $a'$  a binary encoding of MBR  $a$
- $S'$  a binary encoding of spatial sequence  $S$ . Each entry  $S'[i]$  is a binary encoding of  $S[i]$ .

**Similarity of Spatial Sequences** Given two spatial sequences  $P$  and  $Q$ , we say  $P$  *matches*  $Q$  if  $P$  and  $Q$  are of the same length and each spatial object of  $P$  is contained by the corresponding spatial object of  $Q$ , i.e.,  $P[i] \subseteq Q[i]$ , for all  $1 \leq i \leq |P|$ . Now, the problem of subsequence matching of spatial objects can be defined as follows:

- We have a collection of  $N$  spatial sequences  $S_1, S_2, \dots, S_n$ , each of potentially different length.

---

<sup>1</sup>  $S^2$ -Tree stands for Spatial Suffix Tree

<sup>2</sup> A recent improvement of suffix trees called String B-Tree [14] achieves  $O(a + \log n)$  in the worst case.

- We have a query subsequence  $\mathcal{Q}$  of length  $|\mathcal{Q}|$ .
- We want to find quickly all the sequences  $S_i$ , along with the correct offset  $k$ , such that the subsequence  $l = S_i[k, k + |\mathcal{Q}| - 1]$  is enveloped by the query sequence, that is, each spatial object in  $l$  is completely inside its corresponding spatial object in  $\mathcal{Q}$ .

When a query subsequence  $\mathcal{Q}$  is given, a *matching tolerance* is specified *implicitly* at the same time. A bigger MBR represents a higher tolerance. The user has the freedom to enlarge/reduce the size of *each* MBR in the query subsequence, i.e., the tolerance can be customized for different portions of the query subsequence. For example, to specify “don’t care”, the user can simply make some MBRs in the query subsequence as large as the universe so that each spatial object in the database matches the MBRs at those positions.

### 3.1 Three Steps to Constructing the $S^2$ -Tree

**Creating a multidimensional index structure** Given a set of spatial sequences, we add all the spatial objects in those sequences into a multidimensional index structure. The following are the major concerns when we choose our spatial access method:

- *Dimensionality.* One design goal of the  $S^2$ -Tree is for indexing the features extracted from databases of different contents. These features can have 3-20 dimensions. The index structure should be able to handle dimensions in this range.
- *Overlap.* Overlap is the percentage of the volume covered by more than one directory MBR. The  $S^2$ -Tree maps a spatial object into binary strings according to the hierarchy of the index structure (detailed discussion follows). Minimizing the overlap is equivalent to minimizing the number of mappings for each spatial object.
- *Space Utilization.* The size of the index structure is another concern. Since the length of the encoded binary string depends on both the width and depth of the tree structure, maximizing storage utilization is equivalent to minimizing the total number and length of the binary strings.

After comparing the  $R^+$ -tree, the  $R^*$ -tree and the X-tree, we chose the X-tree for our index structure. The notion of supernode introduced by the X-tree creates a balance between overlap and space utilization and is considered more suitable for our purpose.

**Encoding the X-tree** The root node of the X-tree is labeled with  $\epsilon$ , the empty string. An edge connecting a node with its  $k^{\text{th}}$  child is labeled  $k$  (in binary,  $k \geq 0$ ). Nodes other than the root are labeled with the concatenation of the labels on the edges connecting the root to that node.

Thus, we have generated an alphabet  $\mathcal{A} = \{\text{labels on all the nodes}\}$ . The following property holds for the prefix relationship among the symbols in the alphabet:

**Theorem 1.** *If  $\alpha, \beta \in \mathcal{A}$  and  $\alpha$  is a prefix of  $\beta$ , then the MBR of the node labeled with  $\alpha$  contains the MBR of the node labeled with  $\beta$ .*

*Proof:* Since  $\alpha$  is a prefix of  $\beta$ , according to the encoding method, the node labeled with  $\alpha$  must be an ancestor of the node labeled with  $\beta$ . The property holds because in the X-tree, the MBR of a child node is contained by the MBR of its parent node.

For instance, we have a spatial sequence  $S = abcdefghijklmnop$ . Each symbol  $(a, b, \dots, p)$  in the sequence is a point in a 2-dimensional space. Figure 1(a) shows these points and Figure 1(b) is

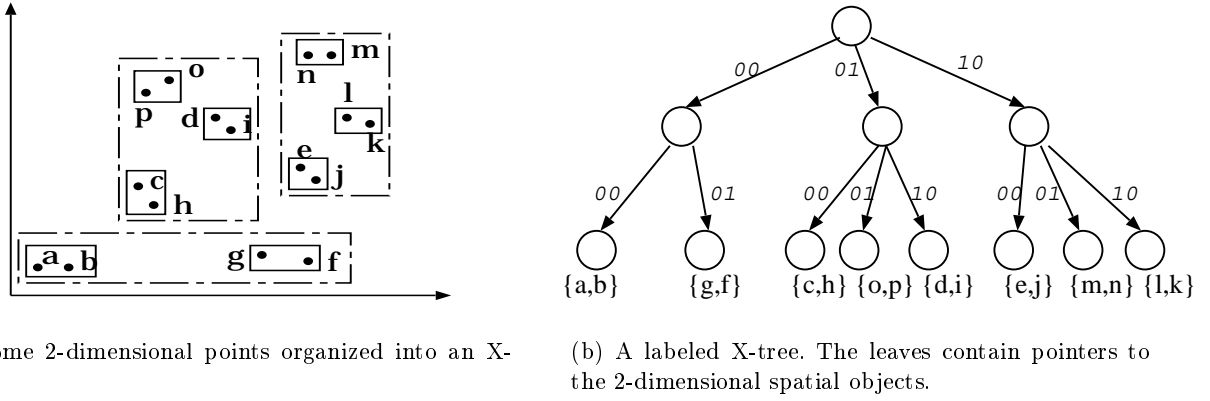


Fig. 1. Using X-tree to cluster spatial objects

the corresponding X-tree built on these points (assuming minimal and maximal number of entries per node are 2 and 3 respectively, and max overlap 20%).

Since the maximum branching factor of the X-tree in Figure 1(b) is 3, we need no more than two bits to label an edge. Each node is coded as the concatenation of the labels on the path from the root to that node. Thus the leftmost leaf node to which points  $a$  and  $b$  belong will be coded as 0000, and the rightmost leaf node to which points  $l$  and  $k$  belong 1010. The alphabet  $\mathcal{A}$ , composed of all the codes of the nodes, is as follows:

$$\mathcal{A} = \{\epsilon, 00, 01, 10, 0000, 0001, 0100, 0101, 0110, 1000, 1001, 1010\}$$

**Creating the suffix tree** Representing each spatial object in the original spatial sequence with the code of the node it belongs to, we transform  $S$  into  $S'$  with characters drawn from alphabet  $\mathcal{A}$  (the '\$' sign at the end of  $S'$  marks the end of the sequence):

$$S' = 0000 \cdot 0000 \cdot 0100 \cdot 0110 \cdot 1000 \cdot 0001 \cdot 0001 \cdot 0100 \cdot 0110 \cdot 1000 \cdot 1010 \cdot 1010 \cdot 1001 \cdot 1001 \cdot 0101 \cdot 0101 \cdot \$$$

or, if we write the binary code in decimal numbers:

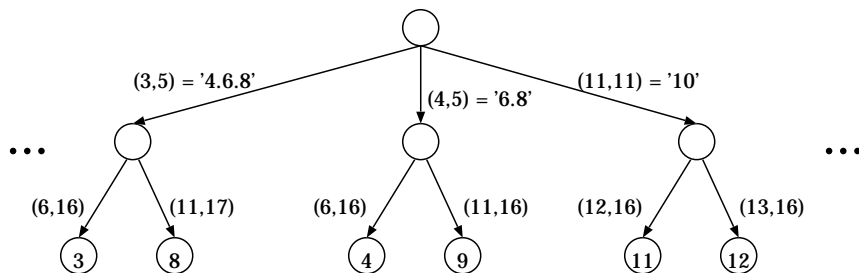
$$S' = 0 \cdot 0 \cdot 4 \cdot 6 \cdot 8 \cdot 1 \cdot 1 \cdot 4 \cdot 6 \cdot 8 \cdot 10 \cdot 10 \cdot 9 \cdot 9 \cdot 5 \cdot 5 \cdot \$$$

We construct a suffix tree in linear time for sequence  $S'$  using McCreight's algorithm[1]. A small part of the suffix tree is shown in Figure 2. The pairs on the edges are the indices in sequence  $S'$ . For instance, (3, 5) represents subsequence  $S'[3, 5]$ , i.e., subsequence 4·6·8 in decimal, or 0100·0110·1000 in binary. The leaves are labeled with the start positions in  $S'$  of the suffixes that they represent. For example, subsequence  $S'[3, 5] = 4 \cdot 6 \cdot 8$  can be found at offset 3 and 8 of  $S'$ .

To construct a suffix tree for a set of spatial sequences, we glue them together into a long sequence by a special symbol '\$', and construct the suffix tree for the concatenated sequence.

### 3.2 Encoding the Query Sequence

Given a query sequence  $\mathcal{Q}$ , we need to encode it into binary strings over alphabet  $\mathcal{A}$  before we can do the search. A spatial object, or its MBR, may correspond to several symbols in the alphabet. We use procedure `EncodeSpatialObject(mbr, root, set)`, where `root` is the root of the X-tree, to encode the `mbr` into a set of symbols. Figure 3 shows the algorithm. It performs a depth-first



**Fig. 2.** The suffix tree built for sequence  $S'$ . Each edge is succinctly represented by a pair of indices in the data structure of the suffix tree.

search of the X-tree, looking for the uppermost internal nodes contained entirely inside  $mbr$ , or leaf nodes that intersect  $mbr$ . For instance, a spatial object which contains the MBR of the root node will be encoded into a single symbol,  $\epsilon$ .

---

```

Procedure EncodeSpatialObject( $mbr$ ,  $Node$ ,  $symbol\_set$ )
Input:           $mbr$  is the MBR of a spatial object,  $Node$  is a node of the X-tree
Input & Output:  $symbol\_set$  is a set of binary characters in alphabet  $\mathcal{A}$ 

01 if  $mbr$  contains the MBR of  $Node$  or  $Node$  is a leaf node then
02     add the label of  $Node$  to  $symbol\_set$ 
03 else
04     for each child node  $c$  of  $Node$  do
05         if the MBR of node  $c$  intersects  $mbr$  then
06             EncodeSpatialObject( $mbr$ ,  $c$ ,  $symbol\_set$ )
07         end if
08     end for
09 end if

```

---

**Fig. 3.** `EncodeSpatialObject()` encodes a spatial object into a set of symbols

`EncodeSpatialObject()` maps each spatial object in the query sequence  $Q$  into a set of symbols, and we get a list of symbol sets,  $L$ , for the entire sequence  $Q$ . Encoding fails if any member of list  $L$  is an empty set. The user then will have to raise the tolerance, i.e., to enlarge the MBR at the corresponding position in the query sequence, in order to find any match in the database.

### 3.3 Subsequences Matching in the $S^2$ -Tree

The search algorithm works as illustrated in Figure 5. Given  $L$  (the list of symbol-sets corresponding to a query sequence), the algorithm performs a depth-first search of the suffix tree. At a certain node, if all the symbol sets have been matched ( $N = 0$ ), then the labels of the leaf nodes that are descendants of that node will be the offsets of the query sequence  $Q$  found in  $S'$ . Otherwise, it calls `Search()` recursively on the sub-nodes whose labels match a prefix of  $L$  (line 07).

In contrast to substring matching in traditional suffix trees, our searching algorithm will traverse *multiple* sub-branches of a node when more than one subsequence on the edges matches a prefix of  $L$ . We use procedure `symbolMatch()` in Figure 4 to determine the compatibility between a list of symbols and a list of symbol-sets (which is generated by `EncodeSpatialObject()` for the query sequence). Instead of exact matching, the partial order in the alphabet is used here to decide

---

```

Procedure symbolMatch(seq, L)
Input:   seq is a sequence of symbols; L is a sequence of sets of symbols, and  $|L| = |seq|$ 
Output:  SUCCESS if matches

01 for i=1 to  $|seq|$  do
02     if none of the symbols in set L[i] is a prefix of the symbol  $seq[i]$ 
03         return FAIL
04     end if
03 end for
04 return SUCCESS

```

---

Fig. 4. symbolMatch() decides whether a label can be matched by a symbol-set

whether the MBR of a symbol is contained inside another MBR, thus allowing matching with a flexible tolerance.

It is easy to prove that the above searching algorithm is *correct*, that is, it never misses qualifying subsequences. This is simply because neither the X-tree nor the suffix tree allows false dismissals. However, the result returned in the `offset_set` by the `Search()` procedure may contain some “false alarms”, and they are discarded in the post-processing step (Section 3.4).

---

```

Procedure Search(Node, L, offset_set)
Input:   Node is a node in the suffix tree; L is a sequence of symbol sets
Output:  offset_set is the set of all the start positions in  $S$  of the subsequence match

01  $N \leftarrow |L|$ 
02 if  $N = 0$  then
03     add all the labels on the leaves that are descendants of Node to offset_set
04 end if
05 for each child node c of Node do
06      $S'(i, j) \leftarrow$  the pair index on the edge linking Node and its child c
07     if symbolMatch( $S'[i, j]$ ,  $L[1, j - i + 1]$ ) = SUCCESS then
08         Search(c,  $L[j - i + 2, N]$ , offset_set)
09     end if
10 end for

```

---

Fig. 5. Search() performs subsequence matching on a suffix tree

### 3.4 Minimizing False Alarms

The `Search()` procedure returns a superset of the qualifying subsequences. To filter out false alarms, we check at each offset returned by `Search()` to see if we have a valid match.

This post-processing step is time-consuming when we have a high percentage of false alarms. False alarms are introduced during the encoding of the query sequence. Suppose  $\alpha$  is one of the symbols in the alphabet into which a spatial object  $s$  in the query sequence is encoded, then  $\alpha$  corresponds to a node  $N$  in the X-tree.

- $N$  is a leaf node. The MBR of node  $N$  contains the MBR of  $s$ . Suppose node  $N$  consists of  $k$  spatial objects, we have the risk of introducing as many as  $k$  false alarms by encoding  $s$  into  $\alpha$  since it is possible that none of them is actually contained by  $s$ .
- $N$  is not a leaf node. No false alarm is introduced by this encoding. The MBR of the spatial object contains the MBR of node  $N$ , which means it contains all the MBRs of the spatial objects in the leaf nodes under node  $N$ .

Thus, the number of false alarms is affected by the number of the spatial objects contained in the leaf nodes of the X-tree. However, reducing the block size of the leaf node will bring the following disadvantages to subsequence matching in the suffix tree:

- Smaller block size means we need more nodes to hold all the spatial objects. This translates to a larger alphabet, and a larger suffix tree.
- Usually, the size of the MBR of the leaf node will decrease if it holds fewer objects. Thus, a spatial object will possibly be encoded into more symbols, and the `symbolMatch()` procedure will find more matchings, which means we need to traverse more sub-branches in the suffix tree.

Hence, we need a balance between the number of entries in the leaf nodes and the potential size of the symbol set. To reduce the size of the symbol set, we prune the X-tree bottom-up (Figure 6). The pruning process picks on level  $h - 1$  a node that has the smallest MBR among all the nodes on level  $h - 1$ , where  $h$  is the height of the X-tree, and removes all its child nodes. Now this node becomes a new leaf node, which represents all the entries in its former child nodes. Thus, we reduce the size of the symbol set by increasing the number of the entries in the leaf node. We repeat this process until the size of the alphabet falls below a threshold value. We will study the relationship between the number of entries in the leaf node and the number of false alarms further in Section 5.

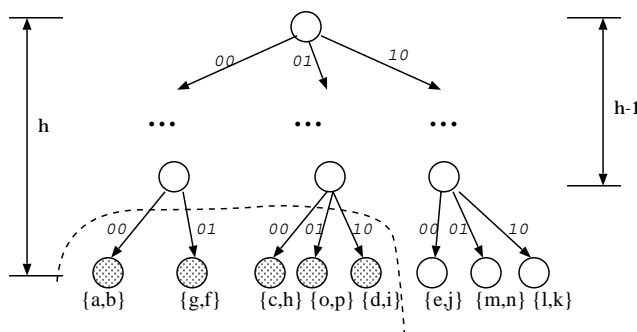


Fig. 6. Pruning the X-tree bottom-up. The parent node now consists of all the spatial objects of its sub-nodes.

### 3.5 Repeated Subsequences

The detection of repeated patterns in sequences is an important activity which crops up in a variety of different situations[17]. However, most similarity-based time series matching algorithms find it difficult to answer queries such as “what is the longest common pattern of the two time series?”

The suffix tree provides the most efficient solution to the longest repeated substring problem[17]. Since the  $S^2$ -Tree is based on the suffix tree, we can use the same technique to solve the longest common pattern problem for spatial sequences.

*Most Frequently Repeated Subsequences* The subsequence represented by a node is repeated  $n$  times if the node has  $n$  leaf nodes, since each leaf node corresponds to an offset in the original sequence.

## 4 Subsequence Matching in Time-Series Databases

Time series databases naturally arise in business as well as scientific decision-support applications. Most current time series subsequence matching algorithms can be seen as consisting of two phases:

1. Converting the time series into sequences of points in multi-dimensional space using DFT or other feature extraction methods.
2. Using some spatial access methods (for example,  $R^*$ -tree) to store and retrieve these features.

For instance, [13] uses DFT to map a time sequence to the frequency domain, drop all but the first few frequencies, and the remaining ones are indexed by the  $R^*$ -tree[8]. This work was generalized by the  $ST$ -index[4] method to allow subsequence matching. The  $ST$ -index uses a sliding window of size  $w$  and place it at every possible position, on every data sequence. For each such placement, features of the subsequence inside the window are extracted. Thus, a data sequence  $S$  is mapped to a trail consisting of  $|S| - w + 1$  points in feature space. The trail is then divided into sub-trails, and the MBRs of the sub-trails are managed by the  $R^*$ -tree.

We noticed several limitations of this pioneering work:

1. The effectiveness of the  $ST$ -index is affected by the length of the query pattern. Since the  $ST$ -index ‘knows’ only about subsequences of length  $w$ , query patterns of length longer than  $w$  will be broken into several sub-queries, each of length  $w$ . It then searches for subsequences that match at least one the sub-queries. This approach will clearly enlarge the searching space.
2. The  $ST$ -index uses a fixed tolerance,  $\epsilon$ , for the entire query pattern. The user might want to have different  $\epsilon$  for different parts. For example, to mark some parts marked as “don’t care”.
3. It is very difficult for the  $ST$ -index to answer queries such as “what are the most frequently repeated sequences of length  $k$ ?” or “what is the longest common pattern?”.
4. The problems of amplitude scaling and offset translation have not been addressed.

To overcome these limitations, we use the  $S^2$ -Tree in phase 2 for subsequence matching of time series data. The  $S^2$ -Tree naturally overcomes the first three limitations mentioned above.

However, in order to solve the problem of amplitude scaling and offset translation, we need an improved feature extraction method in phase 1. In [16], we proposed a new feature extraction algorithm called LANDMARKS, which extracts features that are invariant under certain transformations.

## 5 Experimental Evaluations

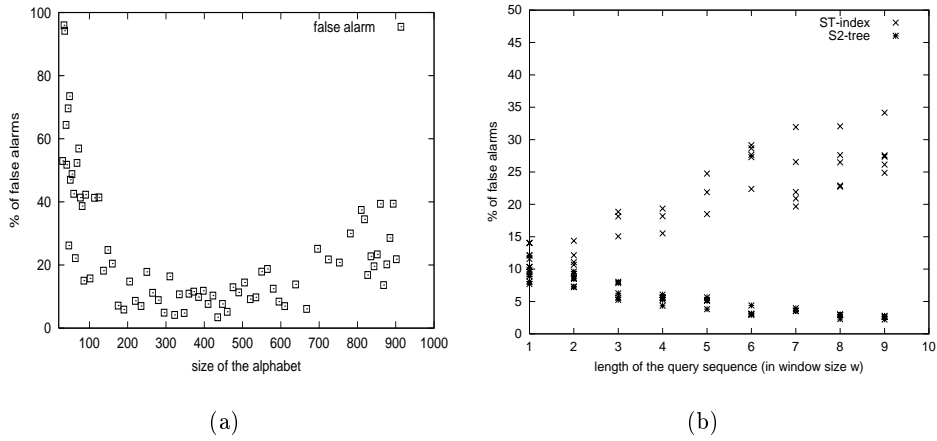
We implemented the  $S^2$ -Tree method and ran our experiments on stock price spreadsheets from Yahoo!. Our environment is a SPARC 20 machine running Solaris 2.7 with 128M memory.

### 5.1 False alarms and the size of the alphabet

We study the relationship between the size of the alphabet and the percentage of false alarms. Figure 7(a) shows that the false alarms drop dramatically when the size of the alphabet increases. However, when the size of the alphabet continues to increase, the percentage of false alarms rebounds. The reason of this phenomenon is explained in Section 3.4. The optimal size of the alphabet depends on both the number and the distribution of the objects in the multidimensional space.

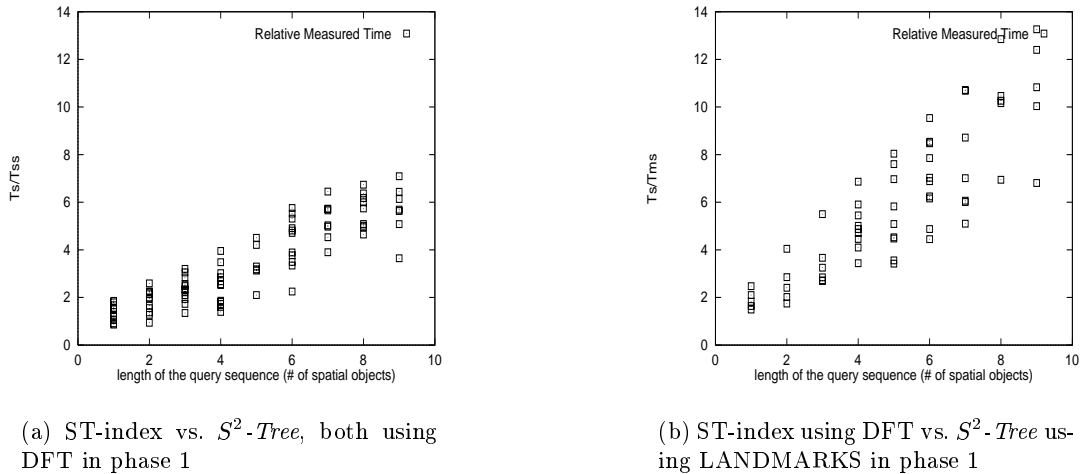
### 5.2 Performance comparison

The use of the  $S^2$ -Tree index structure in phase 2 is independent of the feature extraction methods applied on the time series data in phase 1. The  $ST$ -index uses the first 3 DFT coefficients to map stock prices into a multidimensional space.



**Fig. 7.** (a) Relationship between the size of the encoding alphabet and the percentage of false alarms. The database contains 10,000 spatial objects. (b) Percentage of false alarms varying the length of the query sequence

Figure 7(b) shows the impact of the length of the query sequence on the number of false alarms. These experiments were carried out on spatial points extracted by the DFT with window size  $w = 64$ . For each spatial object in the query sequence, the  $S^2$ -Tree used a universal tolerance  $\epsilon$ , which was also used by the ST-index method in comparison. Since the ST-index method indexes only patterns of length  $w$  – the size of the moving window – the percentage of false alarms increases when the length of the query sequence becomes longer. (For query sequence longer than  $w$ , the ST-index uses the ‘MultiPiece’ algorithm, which searches space volume considerably smaller than the ‘PrefixSearch’ algorithm, but the percentage of false alarm still increases.) To search for a longer query sequence using the  $S^2$ -Tree, however, the percentage of false alarms decreases. This is because when we go deeper in the suffix tree an internal node will have a smaller number of leaf nodes under it.



(a) ST-index vs.  $S^2$ -Tree, both using DFT in phase 1

(b) ST-index using DFT vs.  $S^2$ -Tree using LANDMARKS in phase 1

**Fig. 8.** Relative response time vs. length of query sequences

The  $S^2$ -Tree method outperforms the ST-index for subsequence matching of time-series data. Figure 8(a) shows the relative response time of the ST-index method ( $T_s$ ) vs. the  $S^2$ -Tree method ( $T_{ss}$ ), both using DFT to extract features from the stock prices. The advantage of the  $S^2$ -Tree is also demonstrated by using the LANDMARKS method[16] for feature extraction in Figure 8(b), where  $T_s$  is the response time of the ST-index method, and  $T_{ms}$  that of the  $S^2$ -Tree method using the LANDMARKS for feature extraction. The advantage of the  $S^2$ -Tree method is obvious when the query sequence is mapped to more than one spatial object.

### 5.3 Longest common subsequences of two or more sequences

According to discussions in Section 3.5, the internal nodes of a suffix tree represent the longest common prefixes of suffixes of the input sequences. We can use the  $S^2$ -Tree to search for longest common subsequences of two or more sequences.

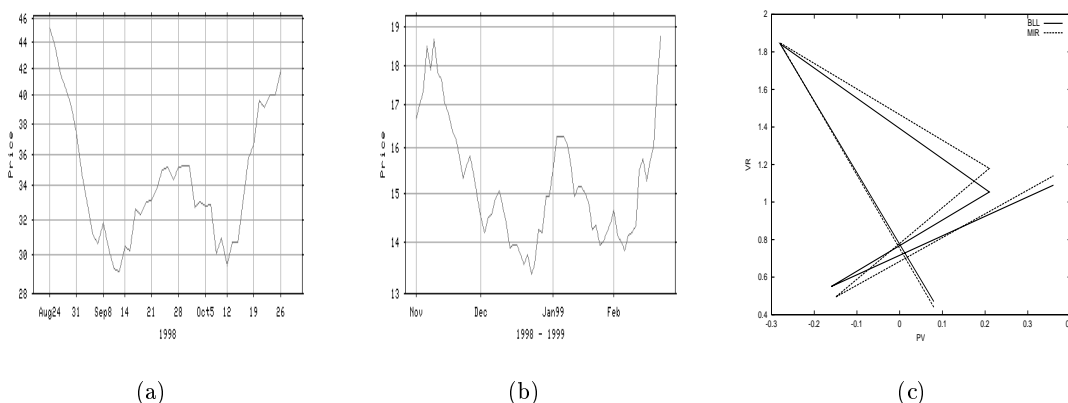


Fig. 9. (a) NYSE:BLL 8/24/98-10/26/98 (b) NYSE:MIR 11/1/98-2/28/99 (c) LANDMARKS features

Figure 9 shows two stock price curves that have the so called “double bottoms” characteristics. The LANDMARKS features of the curves are extracted into a 2-dimensional space:  $pv$  is the percentage of change between the previous landmark and the current one;  $vr$  is the ratio of changes between the previous period and the current one (for detail, see [16]). The  $S^2$ -Tree successfully retrieved the “double bottoms” as the longest common subsequences of the two curves.

## 6 Conclusions

In this paper, we have developed an index method, the  $S^2$ -Tree, for subsequence matching of spatial objects. The insight is the observation that spatial sequences, as well as any other sequences that can be mapped into spatial sequences through feature extraction, are very similar to text strings when it comes to subsequence matching. Thus, we adapt the substring matching techniques, particularly the suffix tree index structure, to subsequence matching of spatial objects. We solved the problem of clustering and encoding spatial objects and most important of all, a partial order that denotes the containment relationship among spatial objects is retained in the encoding. Experiments on

indexing time-series data show that by minimizing false alarms, our algorithm outperforms previous approaches. Also, the  $S^2$ -Tree is capable of locating repeated subsequences and answering queries such as “What is the longest common pattern in the two time-series?”.

## References

1. McCreight, E.M. (1976) “A space-economical suffix tree construction algorithm,” In *Journal of the ACM*, Vol. 23, No.2, pp. 262-72, April 1976.
2. Ukkonen, E. (1992) “Constructing suffix-trees on-line in linear time,” in Leeuwen, J. van(ed.) In *Algorithms, Software, Architecture: Information Processing 92*, Vol. 1, pp. 484-92, Elsevier, Amsterdam.
3. R. Agrawal, K. Lin, H. S. Sawhney, K. Shim: “Fast similarity search in the presence of noise, scaling, and translation in time-series databases,” in *Proceedings of the 21st VLDB Conference*, Switzerland 1995.
4. Christos Faloutsos, M. Ranganathan, Yannis Manolopoulos: “Fast subsequence matching in time-series databases,” In *SIGMOD Conference 1994*: pp. 419-429.
5. S. Berchtold, D. A. Keim, H. Kriegel: “The X-tree: an index structure for high-dimensional data.” In *VLDB 1996*, Mumbai(Bombay), India.
6. Shawney H., Hafner J.: “Efficient color histogram indexing”, In *Proc. Int. Conf. on Image Processing*, 1994.
7. Wallace T., Wintz P.: “An efficient three-dimensional aircraft recognition algorithm using normalized fourier descriptors”, In *Computer Graphics and Image Processing*, Vol. 13, 1980, pp. 99-126.
8. Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: “The R\*-tree: an efficient and robust access method for points and rectangles”, In *SIGMOD 90*, Atlantic City, NJ, 1990, pp. 322-331.
9. Guttman A.: “R-trees: a dynamic index structure for spatial searching”, In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Boston, MA, 1984, pp. 47-57.
10. H. V. Jagadish, R. T. Ng, D. Srivastava: “Substring selectivity estimation”, In *PODS*, Philadelphia, PA, 1999.
11. H. V. Jagadish and A. M. Bruckstein: “On sequential shape descriptions”, In *Pattern Recognition*, 1991.
12. V. Gaede, O. Günther: “Multidimensional access methods”, In *ACM Computing Surveys*, Volumn 30, Number 2, June 1998.
13. R. Agrawal, C. Faloutsos, A. Swami: “Efficient similarity search in sequence databases.” In *Proc. of the Fourth International Conference on Foundations of Data Organization and Algorithms*, Chicago, October 1993.
14. Ferragina P., Grossi R.: “The string B-Tree: a new data structure for string search in external memory and its applications.” In *Journal of the ACM (to appear)*.
15. K. Hinriches, J. Nievergelt: “The grid file: a data structure to support proximity queries on spatial objects.” In *Proc. of the WG'83*, pp. 100-113, Linz, Austria, 1983.
16. C. S. Perng, H. Wang, S. R. Zhang, D. S. Parker: “Landmarks: a new model for similarity-based pattern querying in time series databases.” In *Proc. 16th Int. Conf. on Data Engineering (to appear)*, San Diego, USA, 2000.
17. Graham A. Stephen: *String Searching Algorithms*, Chapter 6, pp. 191-202, World Scientific Publishing Co., 1994.
18. Christos Faloutsos: *Searching Multimedia Databases by Content*, Kluwer Academic Publishers, August 1996.
19. N. J. Ayache and O. D. Faugeras: “HYPER – a new approach for the recognition and position of two-dimensional objects.” In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8, 1986.
20. A. Natsev, R. Rastogi, K. Shim: “WALRUS: a similarity retrieval algorithm for image databases.” In *SIGMOD*, Philadelphia, 1999.
21. D. Vassiliadis: “The input-state space approach to the prediction of auroral geomagnetic activity from solar wind variables.” In *Int. Workshop on App. of Artificial Intelligence in Solar Terrestrial Physics*, Sept. 1993.
22. E. J. Stollnitz, T. D. DeRose, and D. H. Salesin: *Wavelets for Comptuer Graphics: Theory and Applications*. Morgan Kaufmann, 1996.
23. D Rafei, A. Mendelzon: “Efficient retrieval of similar time sequences using DFT.” In *the 5th Intl. Conf. on Foundations of Data Organization (FODO'98)*, Kobe, Japan, November 1998.
24. V. Kobla, D. Doermann, C. Faloutsos: “VideoTrails: representing and visualizing structure in video sequences.” In *ACM Multimedia 1997: 335-346*.
25. Y. Lamdan, H. Wolfson, “Geometric Hashing: A General and Efficient Model Based Recognition Scheme.” In *International Conference on Computer Vision*, 218-249, 1988.
26. Kukich K.: “Techniques for Automatically Correcting Words in Text”, In *ACM Computing Surveys*, Vol. 24, No. 4, 1992, pp. 377-440.
27. Murase H., Nayar S. K: “Three-Dimensional Object Recognition from Appearance-Parametric Eigenspace Method”, In *Systems and Computers in Japan*, Vol. 26, No. 8, 1995, pp. 45-54.
28. Ralf Hartmut Güting, et al: “A Foundation for Representing and Querying Moving Objects”, To appear in *ACM Transactions on Database Systems*.