Fast Subsequence Matching in Time-Series Databases

Christos Faloutsos[*] *M. Ranganathan*[†] *Yannis Manolopoulos*[‡]
Department of Computer Science
and Institute for Systems Research (ISR)
University of Maryland at College Park
email: `christos,ranga,manolopo@cs.umd.edu`

Abstract

We present an efficient indexing method to locate 1-dimensional subsequences within a collection of sequences, such that the subsequences match a given (query) pattern within a specified tolerance. The idea is to map each data sequence into a small set of multidimensional rectangles in feature space. Then, these rectangles can be readily indexed using traditional spatial access methods, like the R*-tree [9]. In more detail, we use a sliding window over the data sequence and extract its features; the result is a trail in feature space. We propose an efficient and effective algorithm to divide such trails into sub-trails, which are subsequently represented by their Minimum Bounding Rectangles (MBRs). We also examine queries of varying lengths, and we show how to handle each case efficiently. We implemented our method and carried out experiments on synthetic and real data (stock price movements). We compared the method to sequential scanning, which is the only obvious competitor. The results were excellent: our method accelerated the search time from 3 times up to 100 times.

1 Introduction

The problem we focus on is the design of fast searching methods that will search a database with time-series of real numbers, to locate subsequences that match a query subsequence, exactly or approximately. Historical, temporal [29] and spatio-temporal [5] databases will benefit from such a facility. Specific applications include the following:

- financial, marketing and production time series, such as stock prices, sales numbers etc. In such databases, typical queries would be '*find companies whose stock prices move similarly*', or '*find other companies that have similar sales patterns with our company*', or '*find cases in the past that resemble last month's sales pattern of our product*'
- scientific databases, with time series of sensor data. For example, in weather data [11], geological, environmental, astrophysics [30] databases, etc., we want to ask queries of the form, e.g., '*find past days in which the solar magnetic wind showed patterns similar to today's pattern*' to help in predictions of the earth's magnetic field [30].

Searching for similar patterns in such databases is essential, because it helps in predictions, hypothesis testing and, in general, in 'data mining' [1, 3, 4] and rule discovery.

For the rest of the paper, we shall use the following notational conventions: If S and Q are two sequences, then:

- $Len(S)$ denotes the length of S
- $S[i:j]$ denotes the subsequence that includes entries in positions i through j
- $S[i]$ denotes the i-th entry of sequence S
- $\mathcal{D}(S,Q)$ denotes the distance of the two (equal length) sequences S and Q.

Similarity queries can been classified into two categories:

- *Whole Matching.* Given a collection of N data sequences of real numbers $S_1, S_2, \ldots, S_N$ and a query sequence Q, we want to find those data sequences that are within distance ϵ from Q. Notice that data and query sequences must have the same length.

*This research was partially funded by the Institute for Systems Research (ISR), by the National Science Foundation under Grants IRI-9205273 and IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc.

†Also with IBM Federal Systems Company, Gaithersburg, MD 20879.

‡On sabbatical leave from the Department of Informatics, Aristotle University, Thessaloniki, Greece 54006.

- *Subsequence Matching.* Given N data sequences $S_1, S_2, \ldots, S_N$ of arbitrary lengths, a query sequence Q and a tolerance ϵ, we want to identify the data sequences S_i $(1 \leq i \leq N)$ that contain matching subsequences (i.e. subsequences with distance $\leq \epsilon$ from Q). Report those data sequences, along with the correct offsets within the data sequences that best match the query sequence. (We assume that we are given a function $\mathcal{D}(S, Q)$, which gives the distance of the sequences S and Q. For example, $\mathcal{D}()$ can be the Euclidean distance.)

The case of 'whole match' queries can be handled as follows [2]: A distance-preserving transform, such as the Discrete Fourier transform (DFT), can be used to extract f features from sequences (eg., the first f DFT coefficients), thus mapping them into points in the f-dimensional feature space. Subsequently, any spatial access method (such as R*-trees) can be used to search for range/approximate queries. This approach exploits the assumption that data sequences and query sequences all have the same length. Here, we generalize the problem and present a method to answer approximate-match queries for subsequences of arbitrary length $Len(Q)$. The ideal method should fulfill the following requirements:

- it should be fast. Sequential scanning and distance calculation at each and every possible offset will be too slow for large databases.
- it should be 'correct'. In other words, it should return all the qualifying subsequences, without missing any (i.e., no 'false dismissals'). Notice that 'false alarms' are acceptable, since they can be discarded easily through a post-processing step.
- the proposed method should require a small space overhead.
- the method should be dynamic. It should be easy to insert and delete sequences, as well as to append new measurements at the end of a given data sequence.
- the method should handle data sequences of varying length, as well as queries of varying length.

The remainder of the paper is organized as follows. Section 2 gives some background material on past related work, on spatial access methods and on the Discrete Fourier Transform. Section 3 focuses on subsequence matching; we propose a new indexing mechanism and we show how off-the-shelf spatial access methods (and specifically the R*-tree) can be used. Section 4 discusses performance results obtained from experiments on real and synthetic data, which show the effectiveness of our method. Section 5 summarizes the contributions of the present paper, giving some extensions of this technique and outlining some open problems.

2 Background

To the best of the authors' knowledge, this is the first work that examines indexing methods for approximate subsequence matching in time-series databases. The following work is related, in different respects:

- indexing in text [13] and DNA databases [6]. Text and DNA strings can be viewed as 1-dimensional sequences; however, they consist of discrete symbols as opposed to continuous numbers, which makes a difference when we do the feature extraction.
- 'whole matching' approximate queries on time-sequences [2] or on color images [14, 21] or even on 3-d MRI brain scans [8]. In all these methods, the idea is to use f feature extraction functions to map a *whole* sequence or image into a point in the (f-dimensional) feature space [18]; then, spatial access methods may be used to search for similar sequences or images. The resulting index that contains points in feature space is called $F-index$ [2].

The F-index works as follows: Given N sequences, all of the same length n, we apply the n-point Discrete Fourier Transform (DFT) and we keep the first few coefficients. Let's assume that we keep f numbers - thus, each sequence is mapped into a point in an f-dimensional space. These points are organized in an R^*-tree, for faster searching. In the typical query, the user specifies a query sequence Q (of length n again) and a tolerance ϵ, requesting all the data sequences that are within distance ϵ from Q. To resolve this query, (a) we apply the n-point DFT on the sequence Q, we keep the f features, thus mapping Q into a f-dimensional point q_f in feature space; (b) we use the F-index to retrieve all the points within distance ϵ from q_f; (c) we discard the false alarms (see more explanations in Lemma 1), and we return the rest to the user.

Here, we generalize the 'F-index' method, which was designed to handle 'whole matching' queries. Our goal is to handle subsequence queries, by mapping data sequences into a few *rectangles* in feature space. Since we rely on spatial access methods as the eventual indexing mechanism, we mention that several multidimensional indexing methods have been proposed, forming three classes: R*-trees [9] and the rest of the R-tree family [15, 17, 28]; linear quadtrees [26, 24]; and grid-files [22].

To guarantee that the 'F-index' method above does not result in any false dismissals, the distance in feature space should match or underestimate the distance between two objects. Mathematically, let O_1 and O_2 be two objects (e.g., same-length sequences) with distance function $D_{object}()$ (e.g., the Euclidean distance) and $F(O_1)$, $F(O_2)$ be their feature vectors (e.g., their first few Fourier coefficients), with distance function $D_{feature}()$ (e.g., the Euclidean distance, again). Then we have:

Lemma 1 *To guarantee no false dismissals for range queries, the feature extraction function $F()$ should satisfy the following formula:*

$$D_{feature}(F(O_1), F(O_2)) \leq D_{object}(O_1, O_2) \quad (1)$$

Proof: Let Q be the query object, O be a qualifying object, and ϵ be the tolerance. We want to prove that if the object O qualifies for the query, then it will be retrieved when we issue a range query on the feature space. That is, we want to prove that

$$D_{object}(Q, O) \leq \epsilon \Rightarrow D_{feature}(F(Q), F(O)) \leq \epsilon \quad (2)$$

However, this is obvious, since

$$D_{feature}(F(Q), F(O)) \leq D_{object}(Q, O) \leq \epsilon \quad (3)$$

□

Following [2], we use the Euclidean distance as the distance function between two sequences, that is, the sum of squared differences. Formally, for two sequences S and Q of the same length l, we define their distance $\mathcal{D}(S, Q)$ as

$$\mathcal{D}(S, Q) \equiv \left(\sum_{i=1}^{l} (S[i] - Q[i])^2 \right)^{1/2} \quad (4)$$

As an example of feature extraction function $F()$ we choose the *Discrete Fourier Transform (DFT)*, for two reasons: (a) it has been used successfully for 'whole matching' [2] and (b) it provides a good, intuitive example to make the presentation more clear. It should be noted that our method is *independent* of the specific feature extraction function $F()$, as long as $F()$ satisfies the condition of Lemma 1 (Eq. 1). If the distance among objects (data sequences) is the Euclidean distance, the condition of Lemma 1 is satisfied, by *any* orthonormal transform, such as, the Discrete Cosine transform (DCT) [31], the wavelet transform [25] etc. Next, we give the definition and some properties of the DFT transformation.

The n-point *Discrete Fourier Transform* [16, 23] of a signal $\vec{x} = [x_i]$, $i = 0, \ldots, n-1$ is defined to be a sequence $\vec{X}$ of n complex numbers X_F, $F = 0, \ldots, n-1$, given by

$$X_F = 1/\sqrt{n} \sum_{i=0}^{n-1} x_i \exp\left(-j2\pi F i/n\right) \quad F = 0, 1, \ldots, n-1 \quad (5)$$

where j is the imaginary unit $j = \sqrt{-1}$. The signal $\vec{x}$ can be recovered by the inverse transform:

$$x_i = 1/\sqrt{n} \sum_{F=0}^{n-1} X_F \exp\left(j2\pi F i/n\right) \quad i = 0, 1, \ldots, n-1 \quad (6)$$

X_F is a complex number (with the exception of X_0, which is a real, if the signal $\vec{x}$ is real). The *energy* $E(\vec{x})$ of a sequence $\vec{x}$ is defined as the sum of energies (squares of the amplitude $|x_i|$) at every point of the sequence:

$$E(\vec{x}) \equiv \| \vec{x} \|^2 \equiv \sum_{i=0}^{n-1} |x_i|^2 \quad (7)$$

A fundamental observation for the correctness of our method is Parseval's theorem [23], which states that the DFT preserves the energy of a signal:

Theorem (Parseval). Let $\vec{X}$ be the Discrete Fourier Transform of the sequence $\vec{x}$. Then we have:

$$\sum_{i=0}^{n-1} |x_i|^2 = \sum_{F=0}^{n-1} |X_F|^2 \quad (8)$$

Since the DFT is a linear transformation [23], Parseval's theorem implies that the DFT also preserves the Euclidean distance between two signals $\vec{x}$ and $\vec{y}$:

$$\mathcal{D}(\vec{x}, \vec{y}) = \mathcal{D}(\vec{X}, \vec{Y}) \quad (9)$$

where $\vec{X}$ and $\vec{Y}$ are Fourier transforms of $\vec{x}$ and $\vec{y}$ respectively.

We keep the first few (2-3) coefficients of the DFT as the features, following the recommendation in [2]. This 'truncation' results in under-estimating the distance of two sequences (because we ignore positive terms from equation 4) and thus it introduces no false dismissals, according to Lemma 1.

The truncation will introduce only false alarms, which, for practical sequences, we expect to be few. The reason is that most real sequences fall in the class of 'colored noise', which has a skewed energy spectrum, of the form $O(F^{(-b)})$. This implies that the first few coefficients contain most of the energy. Thus, the first few coefficients give good estimates of the actual distance of the two sequences. For $b = 1$, we have the *pink noise*, which, according to Birkhoff's theory [27] models signals like musical scores and other works of art. For $b = 2$, we have the *brown noise* (also known as *random walk* or *brownian walk*) which models stock movements and exchange rates (eg., [10, 20]). For $b > 2$ we have the *black noise* whose spectrum is even more skewed than the spectrum of brown noise; black noise models successfully signals like the water level of a river as it varies over time [20].

Symbols	Definitions.
N	Number of data sequences.
S_i	The i-th data sequence ($1 \leq i \leq N$).
$Len(S)$	Length of sequence S.
$S[k]$	The k-the entry of sequence S.
$S[i:j]$	Subsequence of S, including entries in positions i through j.
Q	A query sequence.
w	Minimum query sequence length.
$\mathcal{D}(Q,S)$	(Euclidean) distance between sequences Q and S of equal length.
ϵ	Tolerance (max. acceptable distance).
f	Number of features.
mc	Marginal cost of a point.

Table 1: Summary of Symbols and Definitions

This concludes our discussion on prior work, which concentrated on 'whole match' queries. Next, we describe in detail how we handle the requests for matching subsequences.

3 Proposed Method

Here, we examine the problem of subsequence matching. Specifically, the problem is defined as follows:

- We are given a collection of N sequences of real numbers S_1, S_2, S_N, each one of potentially different length.
- The user specifies query subsequence Q of length $Len(Q)$ (which may vary) and the tolerance ϵ, that is, the maximum acceptable dis-similarity (= distance).
- We want to find quickly all the sequences S_i ($1 \leq i \leq N$), along with the correct offsets k, such that the subsequence $S_i[k : k + Len(Q) \ -1]$ matches the query sequence: $\mathcal{D}(Q, S_i[k : k + Len(Q) \ -1]) \leq \epsilon$.

The brute-force solution is to examine sequentially every possible subsequence of the data sequences for a match. We shall refer to this method by *'SequentialScan'* method. Next, we describe a method that uses a small space overhead, to achieve order of magnitudes savings over the *'SequentialScan'* method. The main symbols used through the paper and their definitions are summarized in Table 1.

3.1 Sketch of the approach - *'ST-index'*

Without loss of generality, we assume that the minimum query length is w, where w (≥ 1) depends on the application. For example, in stock price databases, analysts are interested in weekly or monthly patterns because shorter patterns are susceptible to noise [12]. Notice that we never lose the ability to answer shorter than w queries, because we can always resort to sequential scanning.

Generalizing the reasoning of the method for 'whole matching', we use a sliding window of size w and place it at every possible position (offset), on every data sequence. For each such placement of the window, we extract the features of the subsequence inside the window. Thus, a data sequence of length $Len(S)$ is mapped to a trail in feature space, consisting of $Len(S)$ -w+1 points: one point for each possible offset of the sliding window. Figure 1(a) gives an example of trails. Assume that we have two sequences, S_1 and S_2 (not shown in the figure), and that we keep the first f=2 features (eg, the amplitude of the first and second coefficient of the w-point DFT). When the window of length w is placed at offset=0 on S_1, we obtain the first point of the trail C_1; as the window slides over S_1, we obtain the rest of the points of the trail C_1. The trail C_2 is derived by S_2 in the same manner.

Figure 4 gives an example of trails, using a real time-series (stock-price movements).

One straightforward way to index these trails would be to keep track of the individual points of each trail, storing them in a spatial access method. We call this method *'I-naive'* method, where 'I' stands for 'Index' (as opposed to sequential scanning). When presented with a query of length w and tolerance ϵ, we could extract the features of the query and search the spatial access method for a range query with radius ϵ; the retrieved points would correspond to promising subsequences; after discarding the false alarms (by retrieving all those subsequences and calculating their actual distance from the query) we would have the desired answer set. Notice that the method will not miss any qualifying subsequence, because it satisfies the condition of Lemma 1.

However, storing the individual points of the trail in an R*-tree is inefficient, both in terms of space as well as search speed. The reason is that, almost every point in a data sequence will correspond to a point in the f-dimensional feature space, leading to an index with a 1:f increase in storage requirements. Moreover, the search performance will also suffer because the R-tree will become tall and slow. As we shall see in the section with the experiments, the *'I-naive'* method ended up being almost twice as slow as the *'SequentialScan'*. Thus, we want to improve the *'I-naive'* method, by making the representation of the trails more compact.

The solution we propose exploits the fact that successive points of the trail will probably be similar,

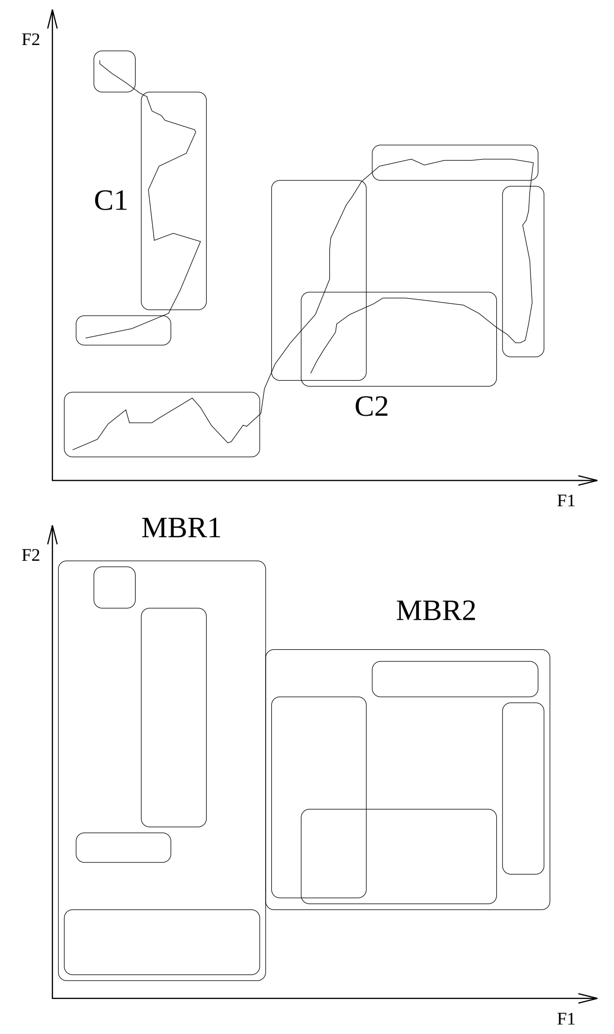


Figure 1: Example of (a) dividing trails into sub-trails and MBRs, and (b) grouping of MBRs in larger ones.

because the contents of the sliding window in nearby offsets will be similar. We propose to divide the trail of a given data sequence into sub-trails and represent each of them with its *minimum bounding (hyper)-rectangle (MBR)*. Thus, instead of storing thousands of points of a given trail, we shall store only a few MBRs. More important, at the same time we still guarantee 'no false dismissals': when a query arrives, we shall retrieve all the MBRs that intersect the query region; thus, we shall retrieve all the qualifying sub-trails, plus some false alarms (sub-trails that do not intersect the query region, while their MBR does).

Figure 1(a) gives an illustration of the proposed approach. Two trails are drawn; the first curve, labeled $C1$ (in the north-west side), has been divided into three sub-trails (and MBRs), whereas the second one, labeled $C2$ (in the south-east side), has been divided in five sub-trails. Notice that it is possible that MBRs belonging to the same trail may overlap, as $C2$ illustrates.

Thus, we propose to map a data sequence into a *set of rectangles* in feature space. This yields significant improvements with respect to space, as well as with respect to response time, as we shall see in section 4. Each MBR corresponds to a whole sub-trail, that is, points in feature space that correspond to successive positionings of the sliding window on the data sequences. For each such MBR we have to store

- t_{start}, t_{end} which are the offsets of the first and last such positionings;
- a unique identifier for the data sequence (*sequence_id*) and
- the extent of the MBR in each dimension ($F1_{low}, F1_{high}, F2_{low}, F2_{high}, \ldots$).

These MBRs can be subsequently stored in a spatial access method. We have used R*-trees [9], in which case these MBRs are recursively grouped into parent MBRs, grandparent MBRs etc. Figure 1(b) shows how the eight leaf-level MBRs of Figure 1(a) will be grouped to form two MBRs at the next higher level, assuming a fanout of 4 (i.e. at most 4 items per non-leaf node). Note that the higher-level MBRs may contain leaf-level MBRs from different data sequences. For example, in Figure 1(b) we remark how the left-side MBR1 contains a part of the south-east curve C2. Figure 2 shows the structure of a leaf node and a non-leaf node. Notice that the non-leaf nodes do not need to carry information about *sequence_id*'s or offsets (t_{start} and t_{end}).

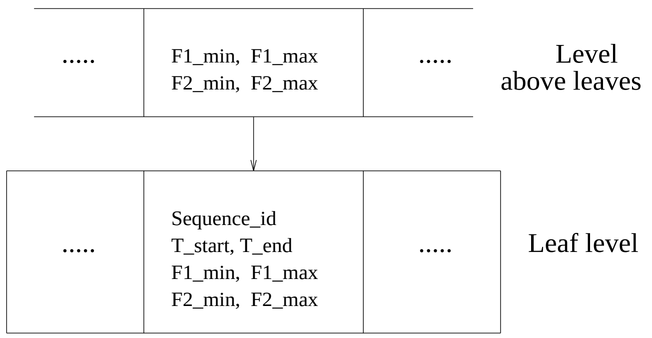


Figure 2: Index node layout for the last two levels.

This completes the discussion of the structure of our proposed index. We shall refer to it by *'ST-index'*, for 'Sub-Trail index'. There are two questions that we have to answer, to complete the description of our method.

- Insertions: when a new data sequence is inserted, what is a good way to divide its trail in feature space into sub-trails.

- Queries: How to handle queries, and especially the ones that are longer than w.

These are the topics of the next two subsections, respectively.

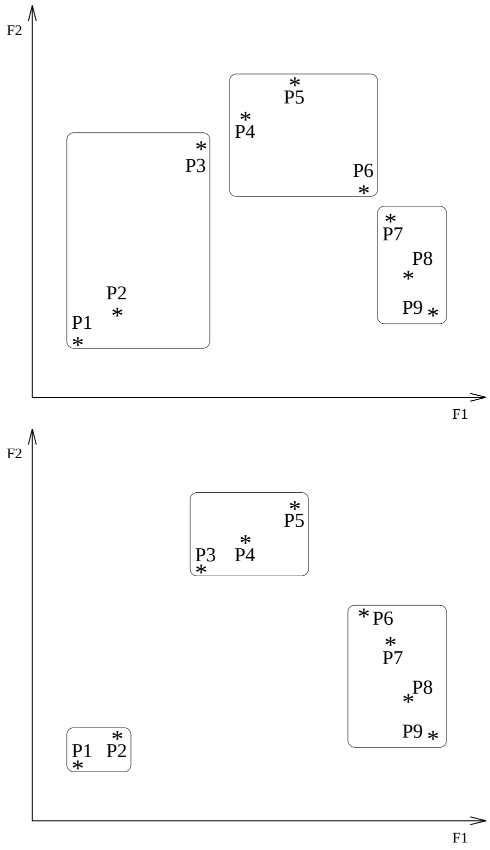


Figure 3: Packing points using (a) a fixed heuristic (sub-trail size = 3), and (b) an adaptive heuristic.

3.2 Insertion - Methods to divide trails into sub-trails

As we saw before, each data sequence is mapped into a 'trail' in feature space. Then the question arising is: how should we optimally divide a trail in feature space into sub-trails and eventually MBRs, so that the number of disk accesses is minimized? A first idea would be to pack points in sub-trails according to a pre-determined, fixed number (e.g., 50). However, there is no justifiable way to decide the optimal value of this constant. Another idea would be to use a simple function of the length of the stored sequence for this sub-trail size (e.g. $\sqrt{Len(S)}$). However, both heuristics may lead to poor results. Figure 3 illustrates the problem of having a pre-determined sub-trail size. It shows a trail with 9 points, and it assumes that the

Method	Description
'SequentialScan'	Sequential scan of the whole database.
'I-naive'	Search using an *'ST-index'* with 1 point per sub-trail.
'I-fixed'	Search using an *'ST-index'* with a fixed number of points per sub-trail.
'I-adaptive'	Search using an *'ST-index'* with a variable number of points per sub-trail.

Table 2: Summary of searching methods and descriptions

sub-trail length is 3 (i.e., = $\sqrt{9}$). The resulting MBRs (Figure 3(a)) are not as good as the MBRs shown in Figure 3(b). We collectively refer to all the above heuristics as the *'I-fixed'* method, because they use an index, with some fixed-length sub-trails. Clearly, the *'I-naive'* method is a special case of the *'I-fixed'* method, when the sub-trail length is set to 1.

Thus we are looking for a method that will be able to adapt to the distribution of the points of the trail. We propose to group points into sub-trails by means of an 'adaptive' heuristic, which is based on a greedy algorithm. The algorithm uses a cost function, which tries to estimate the number of disk accesses for each of the options. The resulting indexing method will be called *'I-adaptive'*. This is the last of the four alternatives we have introduced. Table 2 lists all of them, along with a brief description for each method.

To complete the description of the *'I-adaptive'* method, we have to define a cost function and the concept of *marginal cost of a point*. In [19] we developed a formula which, given the sides $\vec{L} = (L_1, L_2, \ldots L_n)$ of the n-dimensional MBR of a node in an R-tree, estimates the average number of disk accesses $DA(\vec{L})$ that this node will contribute for the average range query:

$$DA(\vec{L}) = \prod_{i=1}^{n}(L_i + 0.5) \quad (10)$$

The formula assumes that the address space has been normalized to the unit hyper-cube ($[0,1)^n$). We use the expected number of disk accesses $DA()$ as the cost function. The marginal cost (mc) of a point is defined as follows: Consider a sub-trail of k points with an MBR of sizes $L_1, \ldots, L_n$. Then the marginal cost of each point in this sub-trail is

$$mc \ = \ DA(\vec{L})/k \quad (11)$$

That is, we divide the cost of this MBR equally among the contained points. The algorithm is then as follows:

```
/* Algorithm Divide-to-Subtrails */
Assign the first point of the trail in a
        (trivial) sub-trail
FOR each successive point
    IF it increases the marginal cost of the
        current sub-trail
    THEN start another sub-trail
    ELSE include it in the current sub-trail
```

3.3 Searching - Queries longer than w

In the previous subsection we discussed how to insert a new data sequence in the *'ST-index'*, using an 'adaptive' heuristic. Here we examine how to search for subsequences that match the query sequence Q within tolerance ϵ. If the query is the shortest allowable ($Len(Q) = w$), the searching algorithm is relatively straightforward:
Algorithm 'Search_Short'

- the query sequence Q is mapped to a point q_f in feature space; the query corresponds to a sphere in feature space with center q_f and radius ϵ;
- we retrieve the sub-trails whose MBRs intersect the query region using our index;
- then, we examine the corresponding subsequences of the data sequences, to discard the false alarms.

Notice that the retrieved MBRs of sub-trails is a *superset* of the sub-trails we should actually retrieve; if a sub-trail intersects the query region, its MBR will definitely do so (while the reverse is not necessarily true). Thus the method introduces no false dismissals.

Handling longer queries ($Len(Q) > w$) is more complicated. The reason is that the *'ST-index'* 'knows' only about subsequences of length w. A straightforward approach would be to select a subsequence (e.g., the prefix) of Q of length w, and use our *'ST-index'* to search for data subsequences that match the prefix of Q within tolerance ϵ. We call this method *'PrefixSearch'*. This will clearly return a superset of the qualifying subsequences: a subsequence T that is within tolerance ϵ of the query sequence Q ($Len(Q) = Len(T)$) will have *all* its (sub)subsequences within tolerance $\leq \epsilon$ from the corresponding subsequence of Q. In general we can prove the following lemma:

Lemma 2 *If two sequences S and Q with the same length l agree within tolerance ϵ, then any pair ($S[i:j]$, $Q[i:j]$) of corresponding subsequences agree within the same tolerance.*

$$\mathcal{D}(S,Q) \leq \epsilon \;\Rightarrow\; \mathcal{D}(S[i:j], Q[i:j]) \leq \epsilon \;\; (1 \leq i \leq j \leq l) \tag{12}$$

Proof: Since $\mathcal{D}()$ is the Euclidean distance, we have

$$\mathcal{D}(S,Q) \leq \epsilon \;\Rightarrow \sum_{k=1}^{l} (S[k]-Q[k])^2 \leq \epsilon^2 \tag{13}$$

Since

$$\sum_{k=i}^{j} (S[k]-Q[k])^2 \leq \sum_{k=1}^{l} (S[k]-Q[k])^2 \tag{14}$$

we have

$$\mathcal{D}(S[i:j], Q[i:j]) = \sum_{k=i}^{j} (S[k]-Q[k])^2 \leq \epsilon \tag{15}$$

which completes the proof. □

Using the *'PrefixSearch'* method, the query region in feature space is a sphere of radius ϵ, and therefore, it has volume proportional to ϵ^f. Next, we show how to reduce the volume of the query region and subsequently, the number of false alarms. Without loss of generality, we assume that $Len(Q)$ is an integral multiple of w; if this is not the case, we use Lemma 2 and keep the longest prefix that is a multiple of w.

$$Len(Q) \;\; = \;\; p\,w \tag{16}$$

Then, we propose to split the longer query into p pieces of length w each, process each sub-query and merge the results of the sub-queries. This approach takes full advantage of our *'ST-index'*. Moreover, as we show, the tolerance specified for each sub-query can be reduced to $\epsilon/\sqrt{p}$. The final result is that the total query volume in feature space is reduced. The following Lemma establishes the correctness of the proposed method. Consider two sequences Q and S of the same length $Len(Q) = Len(S) = p * w$. Consider their p disjoint subsequences $q_i = Q[i*w+1 : (i+1)*w]$ and $s_i = S[i*w+1 : (i+1)*w]$, where $0 \leq i < p-1$.

Lemma 3 *If Q and S agree within tolerance ϵ then at least one of the pairs (s_i, q_i) of corresponding subsequences agree within tolerance $\epsilon/\sqrt{p}$:*

$$\mathcal{D}(Q,S) \leq \epsilon \;\Rightarrow\; \vee_{i=0}^{p-1} \left(\mathcal{D}(q_i, s_i) \leq \epsilon/\sqrt{p}\right) \tag{17}$$

where $\vee$ indicates disjunction.

Proof. By contradiction: If all the pairs of subsequences have distance $> \epsilon/\sqrt{p}$, then, by adding all these distances, the distance of Q and S will be $> \epsilon$, which is a contradiction. More formally, since for $i = 0, \ldots, p-1$

$$\mathcal{D}^2(q_i, s_i) = \sum_{j=i*w+1}^{(i+1)*w} (q_i[j] - s_i[j])^2 \tag{18}$$

we have that

$$\forall i \quad (\mathcal{D}(q_i, s_i) > \epsilon/\sqrt{p}) \quad \Rightarrow \qquad (19)$$

$$\forall i \quad \left(\sum_{j=i*w+1}^{(i+1)*w} (q_i[j] - s_i[j])^2 > \epsilon^2/p \right) \quad \Rightarrow \qquad (20)$$

$$\sum_{j=1}^{p*w} (Q[j] - S[j])^2 > p * t^2/p = \epsilon^2 \qquad (21)$$

or

$$\mathcal{D}(Q, S) > \epsilon \qquad (22)$$

which contradicts the hypothesis. □

The searching algorithm that uses Lemma 3 will be called *'MultiPiece'* search. It works as follows:
Algorithm 'Search_Long (*'MultiPiece'*)'

- the query sequence Q is broken in p sub-queries which correspond to p spheres in feature space with radius $\epsilon/\sqrt{p}$;
- we use our *'ST-index'* to retrieve the sub-trails whose MBRs intersect at least one the sub-query regions;
- then, we examine the corresponding subsequences of the data sequences, to discard the false alarms.

Next, we compare the two search algorithms (*'PrefixSearch'* and *'MultiPiece'*) with respect to the volume they require in feature space. The volume of an f-dimensional sphere of radius ϵ is given by

$$K \ \epsilon^f \qquad (23)$$

where K is a constant ($K = \pi$ for a 2-dimensional space, etc). This is exactly the volume of the *'PrefixSearch'* algorithm. The *'MultiPiece'* algorithm yields p spheres, each of volume proportional to

$$K(\epsilon/\sqrt{p})^f \qquad (24)$$

for a total volume of

$$K * p * \epsilon^f / \sqrt{p^f} = K * \epsilon^f / p^{f/2-1} \qquad (25)$$

This means that the proposed *'MultiPiece'* search method is likely to produce fewer false alarms, and therefore better response time than the *'PrefixSearch'* method, whenever we have $f > 2$ features.

4 Experiments

We implemented the *'ST-index'* method using the 'adaptive' heuristic as described in section 3, and we ran experiments on a stock prices database of 329,000 points, obtained from `sfi.santafe.edu`. Each point was a real number, having a size of 4 bytes. Figure 4(a) shows a sample of 250 points of such a stock price sequence. The system is written in 'C' under AIX, on an IBM RS/6000. Based on the results of [2], we used only the first 3 frequencies of the DFT; thus our feature space has f=6 dimensions (real and imaginary parts of each retained DFT coefficient). Figure 4(b) illustrates a trail in feature space: it is a 2-dimensional 'phase' plot of the amplitude of the 0-th DFT coefficient vs. the amplitude of the 1-st DFT coefficient. Figure 4(c) similarly plots the amplitudes of the 1-st and 2-nd DFT coefficients. For both figures the window size w was 512 points. The smooth evolution of both curves justifies our method to group successive points of the feature space in MBRs. An R*-tree [9] was used to store the MBRs of each sub-trail in feature space.

We carried out experiments to measure the performance of the most promising method: *'I-adaptive'*. For each setting, we asked queries with variable selectivities, and we measured the wall-clock time on a dedicated machine. More specifically, query sequences were generated by taking random offsets into the data sequence and obtaining subsequences of length $Len(Q)$ from those offsets. For each such query sequence, a tolerance ϵ was specified and the query was run with that tolerance. This method was followed in order to eliminate any bias in the results that may be caused by the index structure, which is not uniform at the leaf level. Unless otherwise specified, in all the experiments we used $w = 512$ and $Len(Q) = w$.

We carried out four groups of experiments, as follows:

(a) Comparison of the the proposed *'I-adaptive'* method against the *'I-naive'* method (the method that has sub-trails, each one consisting of one point).

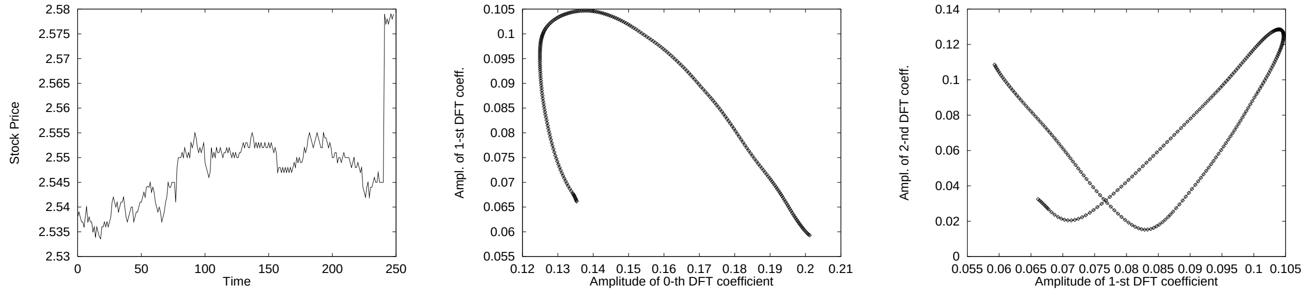
(b) Experiments to compare the response time of our method (*'I-adaptive'*) with sequential scanning for queries of length w.

(c) Experiments with queries longer than w.

(d) Experiments with a larger, synthetic data set, to see whether the superiority of our method holds for other datasets.

Comparison with the *'I-naive'* method. Figure 5 illustrates the index size as a function of the length of the sub-trails for the three alternatives (*'I-naive'* , *'I-fixed'* and *'I-adaptive'*). Our method requires only 5Kb, while the *'I-naive'* method requires ≈24 MB. The *'I-fixed'* method gives varying results, according to the length of its sub-trails.

The large size of the index for the *'I-naive'* method hurts its search performance as well: in our experiments,



(a) (b) (c)

Figure 4: (a) Sample stock-price sequence; (b) its trail in the feature space of the 0-th and 1-st DFT coefficients and (c) the equivalent trail of the 1-st and 2-nd DFT coefficients

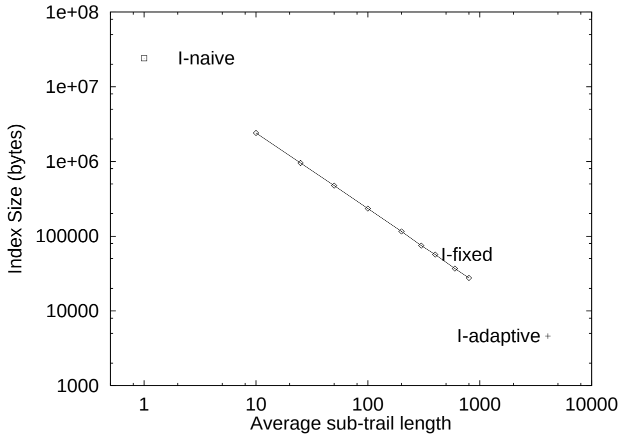


Figure 5: Index space vs. the average sub-trail length (log-log scales)

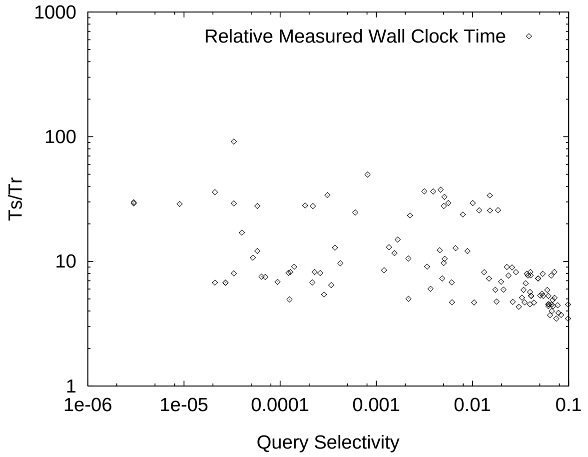


Figure 6: Relative wall clock time vs. selectivity in log-log scale ($Len(Q)$ =w=512 points).

the *'I-naive'* method was approximately *two times* slower than sequentially scanning the entire database.

Response time - 'Short' queries. We start examining our method's response time using the shortest acceptable queries, that is, queries of length equal to the window size ($Len(Q) = w$). We used 512 points for $Len(Q)$ and w. Figure 6 gives the relative response time of the sequential scanning method (T_s) vs. our index assisted method (T_r, where r stands for 'R-tree'), by counting the actual wall-clock time for each method. The horizontal axis is the selectivity; both axes are in logarithmic scales. The query selectivity varies up to 10% which is fairly large in comparison with our 329,000 points time-series database. We see that our method achieves from 3 up to 100 times better response time for selectivities in the range from 10^{-4} to 10%.

We carried out similar experiments for $Len(Q) = w$ = 256, 128 etc., and we observed similar behavior and similar savings. Thus, we omit those plots for brevity. Our conclusion is that our method consistently achieves large savings over the sequential scanning.

Response time - longer queries. Next we examine the relative performance of the two methods for queries that are longer than w. As explained in the previous section, in these cases we have to split the query and merge the results (*'MultiPiece'* method). As illustrated in Figure 7, again the proposed *'I-adaptive'* method outperforms the sequential scanning, from 2 to 40 times.

Synthetic data. In Figure 8 we examine our technique's performance against a time-series database consisting of 500,000 points produced by a random-walk method. These points were generated with a starting value of 1.5, whereas the step increment on each step was ± .001. Again we remark that our method outperforms sequential scanning from 100 to 10 times approximately for selectivities up to 10%.

5 Conclusions

We have presented the detailed design of a method that efficiently handles approximate (and exact) queries for subsequence matching on a stored collection of data sequences. This method generalizes the work in [2],

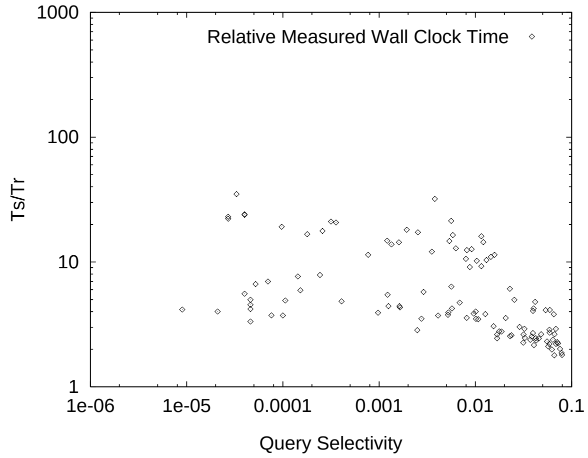


Figure 7: Relative wall clock time vs. selectivity in a log-log scale (w=128, $Len(Q)$ =512 points).

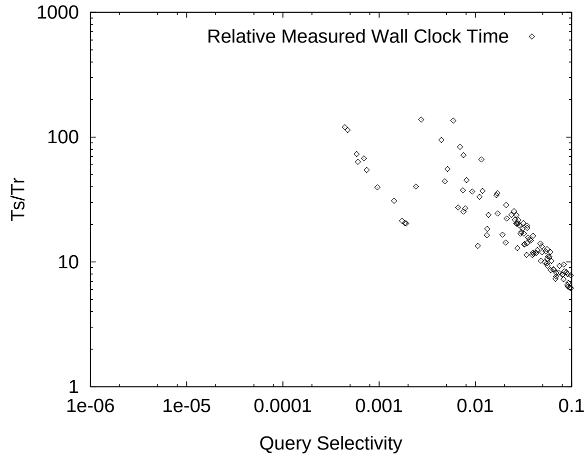


Figure 8: Relative wall clock time vs. selectivity for random walk data in a log-log scale ($Len(Q)$ =w=512 points).

which examined the 'whole matching' case (i.e., all queries and all the sequences in the time-series database had to have the same length). The idea in the proposed method is to map a data sequence into a *set of boxes* in feature space; subsequently, these can be stored in any spatial access method, such as the R*-tree.

The main contribution is that we have designed in detail the first, to our knowledge, indexing method for subsequence matching. The method has the following desirable features:

- it achieves orders of magnitude savings over the sequential scanning, as it was showed by our experiments on real data,
- it requires small space overhead,
- it is dynamic, and
- it is provably 'correct', that is, it never misses qualifying subsequences (Lemmas 1-3)

Notice that the proposed method can be used with *any* set of feature-extraction functions (in addition to DFT), as well as with *any* spatial access method that handles rectangles.

Future work includes the extension of this method for 2-dimensional gray-scale images, and in general for n-dimensional vector-fields (such as 2-d color images to answer queries by image content as in [21], or 3-d MRI brain scans to help detect patterns of brain activation as discussed in [7] etc.)

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *FODO Conference*, Evanston, Illinois, October 1993. also available through anonymous ftp, from olympos.cs.umd.edu:ftp/pub/TechReports/fodo.ps.

[3] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Bala Iyer, and Arun Swami. An interval classifier for database mining applications. *VLDB Conf. Proc.*, pages 560–573, August 1992.

[4] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD*, pages 207–216, May 1993.

[5] K.K. Al-Taha, R.T. Snodgrass, and M.D. Soo. Bibliography on spatiotemporal databases. *ACM SIGMOD Record*, 22(1):59–67, March 1993.

[6] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 1990.

[7] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: a prototype 3-d medical image database system. *IEEE Data Engineering Bulletin*, 16(1):38–42, March 1993.

[8] Manish Arya, William Cody, Christos Faloutsos, Joel Richardson, and Arthur Toga. Qbism: Extending a dbms to support 3d medical images. *Tenth Int. Conf. on Data Engineering (ICDE)*, February 1994. (to appear).

[9] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322–331, May 1990.

[10] Christopher Chatfield. *The Analysis of Time Series: an Introduction.* Chapman and Hall, London & New York, 1984. Third Edition.

[11] Mathematical Committee on Physical and NSF Engineering Sciences. *Grand Challenges: High Performance Computing and Communications.* National Science Foundation, 1992. The FY 1992 U.S. Research and Development Program.

[12] Robert D. Edwards and John Magee. *Technical Analysis of Stock Trends.* John Magee, Springfield, Massachusetts, 1966. 5th Edition, second printing.

[13] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, March 1985.

[14] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 1993. (to appear).

[15] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD*, pages 47–57, June 1984.

[16] Richard Wesley Hamming. *Digital Filters.* Prentice-Hall Signal Processing Series, Englewood Cliffs, N.J., 1977.

[17] H. V. Jagadish. Spatial search with polyhedra. *Proc. Sixth IEEE Int'l Conf. on Data Engineering*, February 1990.

[18] H.V. Jagadish. A retrieval technique for similar shapes. *Proc. ACM SIGMOD Conf.*, pages 208–217, May 1991.

[19] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. *Second Int. Conf. on Information and Knowledge Management (CIKM)*, November 1993.

[20] B. Mandelbrot. *Fractal Geometry of Nature.* W.H. Freeman, New York, 1977.

[21] Wayne Niblack, Ron Barber, Will Equitz, Myron Flickner, Eduardo Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The qbic project: Querying images by content using color, texture and shape. *SPIE 1993 Intl. Symposium on Electronic Imaging: Science and Technology, Conf. 1908, Storage and Retrieval for Image and Video Databases*, February 1993. Also available as IBM Research Report RJ 9203 (81511), Feb. 1, 1993, Computer Science.

[22] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1):38–71, March 1984.

[23] Alan Victor Oppenheim and Ronald W. Schafer. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[24] J. Orenstein. Spatial query processing in an object-oriented database system. *Proc. ACM SIGMOD*, pages 326–336, May 1986.

[25] Mary Beth Ruskai, Gregory Beylkin, Ronald Coifman, Ingrid Daubechies, Stephane Mallat, Yves Meyer, and Louise Raphael. *Wavelets and Their Applications.* Jones and Bartlett Publishers, Boston, MA, 1992.

[26] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[27] Manfred Schroeder. *Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise.* W.H. Freeman and Company, New York, 1991.

[28] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+ tree: a dynamic index for multi-dimensional objects. In *Proc. 13th International Conference on VLDB*, pages 507–518, England,, September 1987. also available as SRC-TR-87-32, UMIACS-TR-87-3, CS-TR-1795.

[29] R. Stam and Richard Snodgrass. A bibliography on temporal databases. *IEEE Bulletin on Data Engineering*, 11(4), December 1988.

[30] Dimitris Vassiliadis. The input-state space approach to the prediction of auroral geomagnetic activity from solar wind variables. *Int. Workshop on Applications of Artificial Intelligence in Solar Terrestrial Physics*, September 1993.

[31] Gregory K. Wallace. The jpeg still picture compression standard. *CACM*, 34(4):31–44, April 1991.