Optimizing Similarity Search for Arbitrary Length Time Series Queries [*][†]

Tamer Kahveci Ambuj Singh

Department of Computer Science

University of California

Santa Barbara, CA 93106

{tamer,ambuj}@cs.ucsb.edu

Abstract

We consider the problem of finding similar patterns in a time sequence. Typical applications of this problem involve large databases consisting of long time sequences of different lengths. Current time sequence search techniques work well for queries of a prespecified length, but not for arbitrary length queries. We propose a novel indexing technique that works well for arbitrary length queries. The proposed technique stores index structures at different resolutions for a given dataset. We prove that, this index structure is superior to existing index structures that use a single resolution. We propose a range query and nearest neighbor query technique on this index structure, and prove the optimality of our index structure for these search techniques. The experimental results show that our method is 4 to 20 times faster than the current techniques including Sequential Scan for range queries, and 3 times faster than Sequential Scan and other techniques for nearest neighbor queries. Because of the need to store information at multiple resolution levels, the storage requirement of our method could potentially be large. In the second part of the paper, we show how the index information can be compressed with minimal information loss. According

[*]A preliminary version of this paper appeared in ICDE 2001 [9].

[†]Work supported partially by NSF under grants EIA-0080134, EIA-9986057, IIS-9877142, and IIS-9817432.

to our experimental results, even after compressing the size of the index to one fifth, the total cost of our method is 3 to 15 times less than the current techniques.

Index Terms: Time series, subsequence search, range query, nearest neighbor query, multiple resolutions.

1 Introduction

Time series or sequence data arises naturally in many real world applications like stock market, weather forecasts, video databases and medical data. Some examples of queries on such datasets include finding companies which have similar profit/loss patterns, finding similar motions in a video database, or finding similar patterns in medical sensor data or biological data. A typical query on such data can be *"Find companies which had a similar profit pattern as company A's profit pattern in January 2000"*, or *"Find companies whose closing prices are similar to company A's profit during the last 6 months"*.

The explosive increase in the size of the time series databases and the variety of time series applications introduces several challenges: 1) The search tools should be able to index and search large databases efficiently. 2) There should not be any artificial restrictions on the lengths of either the database sequence or the query sequence. 3) The search techniques should be scalable with the number of nodes in parallel and distributed architectures.

The need for arbitrary length search arises in most real world sequence database applications for two reasons. First, the database sequences can be of any length. For example, different companies may enter stock market at different times. Therefore, stock histories of different companies can be of any length. Similarly, the length of the video sequences can be of any length from a few minutes to hours. Another striking application is genome databases. The lengths of genome strings vary from a few hundreds to hundreds of millions. Second, the user may be interested in searching query sequences of any length. For example, one can query based on the performance of a company in one month as well as one year. The user may compare a small piece of a video file as well as the whole video file. A biologist may search a short gene string of a few hundred nucleotides or compare the whole chromosome of millions of nucleotides against another.

There are many ways to compare the similarity of two time sequences. One approach is to define the distance between two sequences to be the Euclidean distance in an appropriate multi-dimensional space [1, 4, 6, 11, 19, 25]. Non-Euclidean metrics have also been used to compute the similarity for time sequences. Agrawal, Lin, Sawhney, and Shim [2] use L_∞ as the distance metric. The Landmark model by Perng, Wang, and Zhang [17] chooses only a subset of values from a time sequence, which are peak points, and uses them to represent the corresponding sequence. The authors define distance between two time sequences as a tuple of values, one representing the time and the other the amplitude. In a separate work, Park, Chu, and Hsu [16] use the idea of time warping distance. This distance metric compares sequences of different lengths by stretching them. The distance between two time series data can be made shift and scale invariant by transforming them onto *shift eliminated plane* [10, 5]. Another distance metric D_{norm} is defined by Lee, Chun, Kim, Lee, and Chung [15] for multidimensional sequences. Although this metric has a high recall, it again allows false dismissals, when determining candidate solution intervals. Vlachos, Kollios and Gunopulos [27] proposed to use the Longest Common Subsequence (LCSS) technique in order to find similar trajectories. The authors show that LCSS is more robust to noise than both Euclidean and time warping distances.

The distance between two time series data can be made shift and scale invariant by transforming them onto *shift eliminated plane* [10, 5].

Range searches and nearest neighbor searches in *whole matching* and *subsequence matching* have been the principal queries of interest for time series data. Whole matching corresponds to the case when the query sequence and the sequences in the database have the same length. Agrawal, Faloutsos, and Swami [1] developed the first solution to this problem. They transformed the time sequence to the frequency domain by using DFT (Discrete Fourier Transform). Later, they reduced the number of dimensions to a feasible size by considering the first few frequency coefficients. Chan and Fu [4] used Haar wavelet transform to reduce the number of dimensions and compared this method to DFT. They found that Haar wavelet transform performs better than DFT. However, the performance of DFT can be improved using the symmetry of Fourier Transforms [20, 29]. In this case, both methods give similar results. Wang and Wang [28] proposed to use B-spline wavelet transforms and the least square method to approximate time series data. However, this technique may result in false dismissals.

Subsequence matching is a more difficult problem. Here, the query sequence is potentially shorter than the sequences in the database. The user asks for subsequences in the database that have the same length as the query sequence and a similar pattern. For example, one can ask a query: *Find companies which had a similar profit pattern as company A's profit pattern in January 2000*. A brute force solution is to check all possible subsequences of the given length. However, this is not feasible because of the large number of long sequences in the database.

Faloutsos, Ranganathan, and Manolopoulos [6] proposed *I-adaptive* index to solve the matching problem for queries of prespecified length. They store the trails of the prespecified length in MBRs (Minimum Bounding Rectangles). In the same paper, they also present two methods, *Prefix Search* and *Multipiece Search*, to relax the restriction of prespecified query lengths. Prefix Search performs a database search using a prefix of a prespecified length of the query sequence. Multipiece Search splits the query sequence to non-overlapping subsequences of prespecified length and performs queries for each of these subsequences.

Keogh, Chakrabarti, Pazzani and Mehrotra [13] proposed to split the time sequence into equal sized windows. The average of the values in each window is used to represent all the entries in the window. This compression technique is called the PAA (Piecewise Aggregate Approximation) technique. The experimental results in this paper show that PAA can be computed faster than SVD, Haar and DFT, and the pruning power of PAA is similar to that of DFT and Haar, but worse than SVD. Although the main intent of this paper is to introduce a new dimensionality reduction technique, the authors also propose to perform subsequence searching by sequentially sliding the query sequence over the whole database sequences. The pruning power of the PAA technique can be improved by splitting the time sequences into varying size windows [12]. This technique is called APCA (Adaptive Piecewise Constant Approximation). In a later paper, Popivanov and Miller [18] show that PAA is worse than DFT, Haar wavelets, and Daubechies wavelets. According to these results, Daubechies (DB) wavelets result in the highest precision and the lowest number of page accesses. This can be used to infer that there is no clear winner among all dimensionality reduction techniques, and the benefits of a specific technique depends highly on the data sequences. We use DFT and some wavelet base functions in our experiments.

There are several ways to measure the quality of an index structure. The size of an index structure must be small enough to fit into memory. Furthermore, the running time of the search algorithm must be small. Two parameters are crucial to the running time: precision and the number of disk page accesses. Precision is defined as the ratio of the number of actual solutions to the number of candidate solutions generated by the search algorithm. A good indexing method and a search algorithm should have a high precision while reading few disk pages.

In this paper, we investigate the problem of range and nearest neighbor searching for arbitrary length queries. We propose an optimal index structure and search methods to solve this problem efficiently. Our index structure stores MBRs corresponding to database sequences at different resolutions. To obtain the MBRs of appropriate dimensionality at different resolutions, we consider different compression techniques, namely DFT, Haar and DB2 (Daubechies wavelet). We prove the superiority of this multi-resolution index structure to index structures based on a single resolution like the *I-adaptive* index [6]. The resolutions that optimize the index structure performance in terms of precision and search cost are also derived theoretically.

The range search algorithm splits a given query into several non-overlapping subqueries at various resolutions. For each subquery, the method performs a search on the index structure corresponding to the resolution of the subquery. The results of a subquery are used to refine the radius of the next subquery. We prove the optimality of this search algorithm on our index structure. Our experimental results show that our method is 4 to 20 times faster than the current techniques including Sequential Scan.

The k-nearest neighbor search algorithm works in two phases. The first phase finds an upper bound to the distance of the query to the k^{th} nearest neighbor by performing a search on the index structure. The second phase is a range query using the upper bound found in the first phase followed by a postprocessing.

Storing information at different resolutions increases the amount of storage needed by our index structure. However, a good index structure must not occupy too much space. So, in the second part of the paper we propose two methods to compress the index information. These methods decrease the number of MBRs by replacing a set of MBRs with a larger EMBR (Extended MBR) that tightly covers these MBRs. The first of these methods splits the set of all MBRs into non-overlapping subsets of equal size and replaces each of these subsets with an EMBR. The second method combines MBRs so that the volume of the resulting EMBR is minimized. Our experimental results show that our index structure is 3 to 15 times faster than the current techniques even after compressing the index to one-fifth.

The rest of the paper is as follows. The problem of range queries for subsequence search and the existing techniques are discussed in Section 2. Our index structure and the search algorithms are presented in Section 3. Experimental results are given in Section 4. Index compression methods are presented in Section 5. We end with a brief discussion in Section 6.

2 Current search techniques

Current subsequence search techniques can be classified in two groups based on whether they handle fixed length queries or variable length queries. Section 2.1 discusses the case when the length of the queries is prespecified. Sections 2.2 and 2.3 discuss two solutions for variable length queries.

2.1 Fixed length queries

A simpler version of the subsequence matching problem is when the length of a query sequence equals a prespecified window size w. An indexing method called *I-adaptive* was proposed by Faloutsos et al. [6] for this problem. This method first transforms all possible subsequences of length w in the database using DFT. Only the first few frequency coefficients are maintained after this transformation. An MBR that covers the first subsequence in the frequency domain is created. This MBR is extended to cover the next subsequence if the marginal cost does not increase. Otherwise, a new MBR that covers the new subsequence is created. The resulting MBRs are stored using an R-tree [3, 7].

2.2 Variable length queries: Prefix Search

For variable length queries, Faloutsos et al. propose two different solutions using *I-adaptive* index, namely *Prefix Search* and *Multipiece Search* [6].

Prefix Search assumes that there is a prespecified lower bound w on the length of the query. The method constructs the index structure using *I-adaptive* technique with window size w. Given a query q of an arbitrary length, it searches the database using a length w prefix of q. If the distance between the query q and a subsequence s of length $|q|$ is $d(q, s)$, the Euclidean distance between sequences q and s, then the distance between any of their prefixes can not be greater than $d(q, s)$. Therefore, the method does not cause any false dismissals.

A shortcoming of the Prefix Search method is that its search volume can become unnecessarily large leading to a large number of false hits. This is captured in the following lemma. The Euclidean distance between two sequences x and y is denoted by $d(x, y)$, and the expected value of this distance is represented by $E[d(x, y)]$.

Lemma 1 *Let q be a query sequence and x be a database sequence of the same length. Let q_1 be a subsequence of q, and x_1 be a subsequence of x such that $|q_1| = |x_1|$. Then,*

$$E[d(q, x)] \approx \sqrt{\tfrac{|q|}{|q_1|}}.E[d(q_1, x_1)].$$

Proof:

Let $|q| = |x| = n$. Let $Y_1, Y_2, ..., Y_n$ be iid (independent and identical) random variables, where $Y_i = |q[i] - x[i]|$ for $1 \leq i \leq n$. Let Y be a random variable, whose distribution is equal to Y_i, for $1 \leq i \leq n$. Let $\mu(Y^2)$ and $\sigma^2(Y^2)$ be the mean and the variance of Y^2 respectively.

Define $Z_k = \sum_{i=1}^{k} Y_i^2$, where k is a positive integer. From the Central Limit Theorem, we know that regardless of the distribution of Y^2, for large k, Z_k is normally distributed with mean $\mu(Z_k) = k \cdot \mu(Y^2)$ and variance $\sigma^2(Z_k) = k \cdot \sigma^2(Y^2)$. The probability density function of Z_k can be written as $f_{Z_k}(t) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(t-\mu(Z_k))^2}{2\cdot\sigma^2(Z_k)}}$.

Let T_k be the random variable denoting the Euclidean distance between two randomly chosen time series of length k, then $T_k = \sqrt{Z_k}$. The mean of T_k can be calculated as

$$\mu(T_k) = \int_0^\infty t \cdot f_{Z_k}(t^2) \cdot dt,$$

Then, we have

$$\mu(T_k) \;= \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \int_0^\infty t \cdot e^{-\frac{(t^2-\mu(Z_k))^2}{2\cdot\sigma^2(Z_k)}} \cdot dt.$$

After changing the variables as $u = \frac{t^2-\mu(Z_k)}{\sqrt{2\cdot\sigma^2(Z_k)}}$, we get

$$\begin{aligned}\mu(T_k) &= \frac{1}{2\cdot\sqrt{\pi}} \cdot \int_{\frac{-\mu(Z_k)}{\sigma(Z_k)\cdot\sqrt{2}}}^{\infty} e^{-u^2} \cdot du \\ &= \frac{1}{2\cdot\sqrt{\pi}} \cdot \int_{-\infty}^{\frac{\mu(Z_k)}{\sigma(Z_k)\cdot\sqrt{2}}} e^{-u^2} \cdot du.\end{aligned}$$

It is well known that

$\int_{-\infty}^0 e^{-u^2} \cdot du = \frac{\sqrt{\pi}}{2}$, and the Taylor expansion of the following integral is

$\int_0^t e^{-u^2} \cdot du = \sum_{i=0}^{\infty} (-1)^i \cdot \frac{t^{2\cdot i+1}}{(2\cdot i+1)\cdot i!}$.

From Taylor expansion, we have

$$\begin{aligned}\mu(T_k) &\approx \frac{1}{4} \cdot (1 + \frac{\mu(Z_k)}{\sigma(Z_k)\cdot\sqrt{2}}) \\ &= \frac{1}{4} \cdot (1 + \frac{k\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2\cdot k}}) \\ &= \frac{1}{4} \cdot (1 + \frac{\sqrt{k}\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2}}).\end{aligned}$$

Hence

$$\frac{\mu(T_n)}{\mu(T_k)} = \frac{1+\frac{\sqrt{n}\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2}}}{1+\frac{\sqrt{k}\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2}}}.$$

For large values of n and k, we can ignore 1's, then we have

$$\begin{aligned}\frac{\mu(T_n)}{\mu(T_k)} &\approx \frac{\frac{\sqrt{n}\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2}}}{\frac{\sqrt{k}\cdot\mu(Y^2)}{\sigma(Y^2)\cdot\sqrt{2}}} \\ &= \sqrt{\frac{n}{k}}.\end{aligned}$$

Hence we conclude that

$$E[d(q,x)] \approx \sqrt{\frac{|q|}{|q_1|}}.E[d(q_1,x1)]. \quad \square$$

As a consequence of the above lemma, if $d(q_1, x_1) = r$ then $d(q,x) \approx r \cdot \sqrt{\frac{|q|}{|q_1|}}$. As a result, searching for r-distant prefixes using q_1 is tantamount to searching for $r \cdot \sqrt{\frac{|q|}{|q_1|}}$-distant sequences using q. We refer to the radius $r \cdot \sqrt{\frac{|q|}{|q_1|}}$ as the *implicit search radius*, and to the corresponding search volume as the *implicit search volume*.

If the Prefix Search method uses dim dimensions for the transformed sequences, the implicit search volume increases by a factor of $\sqrt{\frac{|q|}{w}}^{dim}$. If $|q|$ is much larger than the window size $w = |q_1|$, the method incurs many false retrievals. This decreases the precision and increases the number of disk reads. For example, let the length of a query sequence be 16 times the window size. In this case, the implicit search radius will be 4 times the actual search radius. If the index stores 6 dimensions, then the implicit search volume will be more than 4000 times the actual search volume.

2.3 Variable length queries: Multipiece Search

Multipiece Search assumes that the length of the query sequence is an integer multiple of the window size[1], i.e. $|q| = kw$ for some integer k. Given a query sequence q and a radius ϵ, the method divides the query sequence into k non-overlapping subsequences of length w. Later, the method runs k subqueries, i.e. one for each subsequence, with radius $\epsilon/\sqrt{k}$, and combines the results of these subqueries. The method does not incur any false dismissals, because if a sequence is within ϵ distance of the query, then at least one of the corresponding subsequences must be within the distance $\epsilon/\sqrt{k}$.

However, there are several problems with the Multipiece Search technique. 1) The cost of a query increases linearly with its size. The radius for each subquery is $\epsilon/\sqrt{k}$. From Lemma 1, the likelihood of finding a subsequence of length w within a distance of $\epsilon/\sqrt{k}$ from a subquery is the same as that of finding a sequence of length kw within a distance of ϵ from the original query. This implies that the expected cost of each subquery is the same as that of the original query. The number of subqueries (k) increases linearly with the size of the original query. As a result, the total search cost of the Multipiece Search technique increases linearly with the length of the query. 2) Multipiece Search has a postprocessing step, in which the results from different subqueries are combined and filtered. This has an extra CPU overhead.

[1]If the query length is not an exact multiple of the window size, then the longest prefix that is a multiple of the window size can be used.

3 Proposed method

The main problem with the two solutions for variable length queries is that the index structure does not use all the information available in the query sequence, due to the static structure of the index and the unpredictable length of queries. Our solution alleviates this problem by storing information at different resolutions in the database.

3.1 Storing information at multiple resolutions

Let s be the longest time sequence in the database, where $2^b \leq |s| < 2^{b+1}$ for some integer b. Similarly, let the minimum possible length for a query be 2^a, for some integer a where $a \leq b$. Let s_1, s_2, ..., s_n be the time sequences in the database as shown in Figure 1. Our index structure stores a grid of trees $T_{i,j}$, where i ranges from a to b, and j ranges from 1 to n. Tree $T_{i,j}$ is the set of MBRs for the j^{th} sequence corresponding to window size 2^i. In order to obtain $T_{i,j}$, we transform each sequence of length 2^j in sequence s_i using DFT or wavelets, and choose a few of the coefficients from the transformation. The transformed sequences are stored in MBRs similar to the *I-adaptive* index structure. We begin with an initial MBR. It is extended to cover the next subsequence of length w until the marginal cost of the MBR increases. When the marginal cost of an MBR increases, a new MBR is created and used for later subsequences. The i^{th} row of our index structure is represented by R_i, where $R_i = \{T_{i,1}$, ...,$T_{i,n}\}$ corresponds to the set of all trees at resolution 2^i. Similarly, the j^{th} column of our index structure is represented by C_j, where $C_j = \{T_{a,j}$, ...,$T_{b,j}\}$ corresponds to the set of all trees for the j^{th} time sequence in the database. We call this index structure the *MR (Multi Resolution) index structure*. One can consider using a different base (e.g. base 3) to construct the index structure, however base 2 turns out to be the optimal (as shown

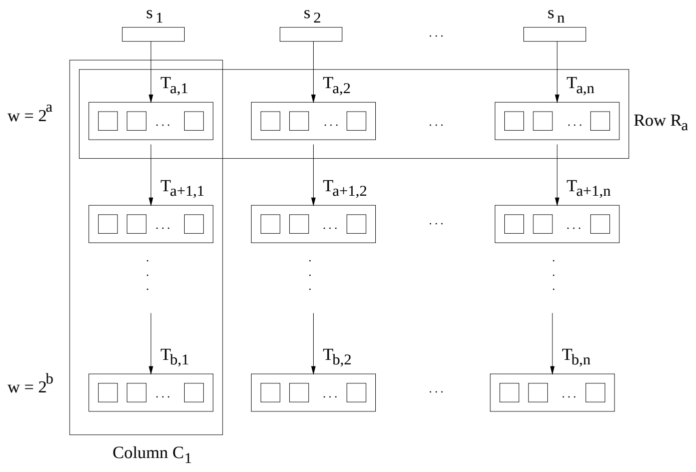


Figure 1. Layout of the index structure.

later). An MR index structure that uses a base of i is referred to as a *base-i MR index structure*.

If each MBR of the index structure contains c points on the average, then the size of $R_i = 2 \cdot d \cdot \sum_{j=1}^{n} \frac{|s_j|-2^i+1}{c}$, where d is the number of dimensions of the index structure. This is because each time sequence s_j corresponds to $\frac{|s_j|-2^i+1}{c}$ MBRs, and each MBR stores two vectors (one for lower coordinates and one for higher coordinates) of d dimensions. Based on this formula, we conclude that the size of R_i decreases as i (i.e. resolution) increases. However, if the time sequences are very long compared to the resolutions of the index structure then the size of the rows decreases slowly. The total space complexity of the index structure is $O(a \cdot D/c)$, where a is the number of different resolution and D is the database size. In Section 5 we will discuss how the index structure can be compressed.

The MR index structure can be constructed by sequentially scanning the database only once. Therefore, the time complexity of index construction is $O(D)$.

3.2 Longest Prefix Search (LPS)

Let q_1 be a prefix of a query sequence q. In Lemma 1 we proved that a range query using q_1 with a query radius of ϵ has an implicit search radius of $\epsilon \cdot \sqrt{\frac{|q|}{|q_1|}}$. The implicit search radius decreases as $|q_1|$ increases. Our first search technique, the *Longest prefix search technique* (LPS), uses the longest prefix (q_1) of the query sequence (for which the resolution $|q_1|$ appears in the MR index structure) to perform a range search on the corresponding row ($R_{log_2|q_1|}$) of the MR index structure to find the candidate set. Later, this candidate set is postprocessed to eliminate false retrievals.

In order to optimize the LPS technique, the length of the largest resolution available in the MR index structure must be maximized. As shown in Lemma 2, this implies that base 2 must be used in the construction of the index structure.

Lemma 2 *Let MR$_i, i \geq 2$, be the base-i MR index structure constructed on a time series dataset. If queries whose lengths are uniformly distributed between 1 and N (for some large integer N) are performed on these index structures with equal probability, then the expected length of the longest prefix of these queries is maximized for MR$_2$.*

Proof:

The leading nonzero digit of the base-i representation of $|q|$ corresponds to the longest prefix of q whose resolution is available in MR$_i$. Define $h = log_i N$. Let us partition the numbers $1 \dots N$ based on the number of leading zeros in the h digit base-i representation. Let P_k, $0 \leq k \leq h$, denote the partition corresponding to k leading zeros. All the numbers in partition P_k use a prefix of length i^{h-k-1} for range searching in the LPS technique. The size of partition P_k is $(i-1) \cdot i^{h-k-1}$. The leading nonzero digit can take $i-1$ values and the remaining digits can take i values each. Therefore, the average length of the longest prefix for MR$_i$ is:

$$
\begin{aligned}
L_i &= \tfrac{1}{N} \cdot \textstyle\sum_{k=1}^{h-1} (i-1) \cdot i^{h-k-1} \cdot i^{h-k-1}. \\
&= \tfrac{i-1}{N} \cdot \textstyle\sum_{k=1}^{h-1} (i^2)^{h-k-1} \\
&= \tfrac{i-1}{N} \cdot \tfrac{i^{2h}-1}{i^2-1} \\
&= \tfrac{N^2-1}{N \cdot (i+1)} \\
&\simeq \tfrac{N}{i+1}
\end{aligned}
$$

Therefore, the length of the longest prefix decreases as the base of the MR index structure increases. □

Lemma 2 proves that using powers of 2 for the resolutions of the MR index structure maximizes the performance of the LPS technique. A base-2 MR index structure guarantees that the implicit search radius for any query does not increase by a factor of more than $\sqrt{2}$.

3.3 An improved search algorithm

The LPS technique uses the longest prefix of the query sequence whose resolution is available in the MR index structure. Although this technique is better than the Prefix Search technique (i.e. the implicit search radius is less), it still does not use all the information in the query sequence. Our second search technique partitions a given query sequence of arbitrary length into a number of subqueries at various resolutions available in our index structure. Later, it performs a *partial range query* for each of these subqueries on the corresponding row of the index structure. This is called a partial range query, because it only computes distance of the query subsequence to the MBRs, not the distance to the actual subsequences contained in the MBRs.

Given any query q of length $k \cdot 2^a$ and a range ϵ, there is a unique partitioning[2], $q = q_1 q_2 ... q_t$, with $|q_i| = 2^{c_i}$ and $a \le c_1 < ... < c_i \le c_{i+1} \le ... c_t \le b$. This partitioning corresponds to the 1's in the binary representation of k. We first perform a search using q_1 on row R_{c_1} of the index structure. As a result of this search, we obtain a set of MBRs that lie within a distance of ϵ from q_1. Using the distances to these MBRs, we refine the value of ϵ for each MBR and make a second query using q_2 on row R_{c_2} and the new value of ϵ. This process continues for the remaining rows R_{c_3} ... R_{c_t}. The idea of the above radius refinement is captured in the following lemma.

[2]Similar to the Multipiece Search technique, if the length of the query sequence is not a multiple of the minimum window size, then the longest prefix of the query whose length is a multiple of the minimum window size can be used.

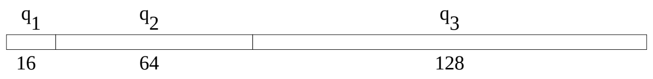


Figure 2. Partitioning for query q, $|q|$ = 208.

Lemma 3 *Let q be a given query sequence and x be an arbitrary subsequence of the same length in the database. Let q_1 be a prefix of q and q_2 be the rest of q. Similarly, let x_1 be a prefix of x and x_2 be the rest of x, such that $|q_1| = |x_1|$. If $d(q, x) \leq \epsilon$ then*

$$d(q_2, x_2) \leq max_{B \in \mathcal{B}}(\sqrt{\epsilon^2 - d(q_1, B)^2}),$$

where B is an MBR that covers x_1 and $\mathcal{B}$ is any arbitrary set of rectangles that contains B.

Proof:

Since $d(q, x) = \sqrt{d(q_1, x_1)^2 + d(q_2, x_2)^2} \leq \epsilon$, we have

$d(q_2, x_2) \leq \sqrt{\epsilon^2 - d(q_1, x_1)^2}$. Since $d(q_1, x_1) \geq d(q_1, B)$,

$d(q_2, x_2) \leq \sqrt{\epsilon^2 - d(q_1, B)^2})$. From this, it follows that

$$d(q_2, x_2) \leq max_{B \in \mathcal{B}}(\sqrt{\epsilon^2 - d(q_1, B)^2}). \quad \square$$

An example partitioning of a query sequence is shown in Figure 2. In this example, the length of the query sequence is 208 and the minimal query length is 16. The query is split into 3 pieces of length 16, 64 and 128. Three subqueries are performed, one for each partition.

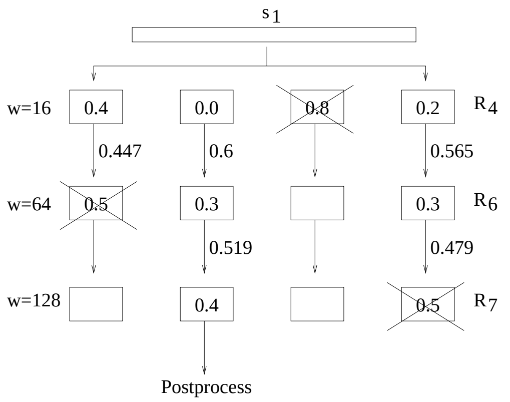


Figure 3. The execution of a sample range query on database sequence s_1 with $|q|$ = 208, and ϵ = 0.6. The real number displayed inside each MBR represents the distance between the corresponding query subsequence and that MBR. The real numbers next to each arrow show the refined radius after processing that MBR.

Figure 3 depicts the execution of the range query algorithm on database sequence s_1 for $\epsilon = 0.6$ for the query sequence in Figure 2. In this example, s_1 contains four MBRs for each resolution. The real number displayed inside each MBR represents the distance between the corresponding query subsequence and that MBR. The real numbers next to each arrow show the refined radius after processing that MBR. The first subquery q_1 is performed on row R_4 with ϵ = 0.6 as the radius. As a result of this search, ϵ is refined to ϵ'_k for each MBR B_k in row R_4. Since the distance between $|q_1|$ and the third MBR is greater than ϵ, the subsequences of s_1 corresponding to this MBR are pruned at this step. The next subquery q_2 is performed on row R_6 with the smaller radius ϵ'_k for each MBR B_k. As a result of this subquery, ϵ'_k is refined further to ϵ''_k. After the second subquery, the subsequences of the first MBR are also pruned since the distance of q_2 to the first MBR is larger than the current threshold for that MBR. Finally, subquery q_3 is performed on row R_7 with radius ϵ''_k for each MBR B_k. The fourth MBR of s_1 is pruned away at this step similarly. The resulting set of MBRs (i.e. only the second MBR in this example) is then processed to find the actual subsequences using disk I/Os. The same algorithm is run for other database sequences as well.

Figure 4 presents the complete search algorithm. Step 1 partitions the query q into separate pieces corresponding to a subset of the rows R_{c_1}, R_{c_2}, ..., R_{c_t} of the index structure. In Step 2, these rows are searched from top to bottom independently for each sequence (column) in the database. At every row, a partial range query (Step $2b(i)$), and then a range refinement (Step $2b(ii)$) are carried out. When all rows have been searched, the disk pages corresponding to the last result set are read (Step $2c$). Finally postprocessing is carried out to eliminate false retrievals (Step $2d$).

As a consequence of Lemma 3 we have the following theorem.

Theorem 1 *The MR index structure does not incur any false dismissals.*

We note the following about the search algorithm.

1) Every column of the index structure (or each sequence in the database) is searched independently. For each MBR, the refinement of radius is carried out independently using only the results from the related MBRs. For example, if an MBR covers the subsequences starting at positions 300 to 500 for resolution 64 in the previous example (Figure 2), then we only use the results that derive from the MBRs whose starting position ranges intersect with 300-16 to 500-16 for resolution 16.

2) For each sequence, no disk reads are done

```
/*Let $M_j$ be the number of MBRs in row $R_{c_1}$ of database sequence $s_j$ for all $j$.*/
Algorithm RANGE-SEARCH($q, \epsilon$)
  1. Partition the query $q$ into $t$ parts as $q_1$, $q_2$, ..., $q_t$ such that $|q_i| = 2^{c_i}$ and $a \leq c_1 < c_2 < ... < c_t \leq b$.
  2. For $j := 1$ to $n$
     (a) For $k := 1$ to $M_j$
         $\epsilon_{c_1,j}[k] := \epsilon$
     (b) For $i := 1$ to $t$
         i. Perform range query on $T_{c_i,j}$ using $\epsilon_{c_i,j}[k]$. Let $Res_{c_i,j}[k]$ be the resulting set of MBRs which contain the
            subsequences that immediately follow the subsequences in the $k^{th}$ MBR of row $R_{c_i}$, and whose distances
            to $q_i$ are less than $\epsilon_{c_i,j}[k]$.
        ii. $\epsilon_{c_{i+1},j}[k] := max_{B \in Res_{c_i,j}[k]}\{\sqrt{\epsilon_{c_i,j}[k]^2 - d(q_i, B)^2}\}$
     (c) Read disk pages corresponding to $Res_{c_t,j}[k]$ in the candidate set.
     (d) Postprocess to eliminate false retrievals.
```

Figure 4. Range search algorithm.

until the termination of the for loop in Step $2b$. Furthermore, the target pages are read in the same order as their location on disk. As a result, the average cost of a page access is much less than the cost of a random page access.

3) Performing subqueries in increasing length order (i.e. from top to bottom in Figure 1) improves the performance. This is explained in Lemma 4.

4) A special condition occurs when one of the subqueries q_i turns out to be in one of the MBRs (say B) of tree $T_{i,j}$. In this case $d(q_i, B) = 0$, $\epsilon_{c_{i+1},j} = \epsilon_{c_i,j}$, and no radius refinement is possible at this search step. However, if we use the actual subsequences corresponding to B (by accessing disk pages), a refinement may still be possible.

5) One may be tempted to simplify the algorithm by iterating row by row, i.e., by finding MBRs within the current radius for all sequences in a row, and then refining the search radius simultaneously across all sequences for the next row. Such a row based refinement impacts

the performance of the algorithm. This is because the amount of radius refinement is reduced due to non-local considerations. With respect to Lemma 3, this means that a row-based approach increases the size of $\mathcal{B}$, and consequently reduces the amount of refinement.

6) Our range search algorithm can be easily parallelized by splitting the MR index structure vertically. Since the columns of the MR index structure are independent of each other, vertical splitting achieves linear speedup in parallel and distributed environments.

3.4 Theoretical Analysis

In section 3.2 we proved that the base-2 MR index structure maximizes the expected length of the longest prefix of the query sequence, hence maximizes the performance of the LPS technique. In this section, we present a theoretical analysis of the MR index structure for the search technique proposed in Section 3.3. We first show that top-down search on the MR index structure has better expected performance than any other traversal order. Next, we verify that the base-2 MR index structure minimizes the expected number of subqueries, hence minimizes the CPU cost for pre-processing.

3.4.1 Ordering the MR index rows

One can consider various traversal strategies on the MR index structure by performing subqueries in different order. Lemma 4 proves that performing subqueries in increasing length order minimizes the expected search cost.

Lemma 4 *The performance of the MR index structure is optimized if it is traversed from top to bottom (i.e. subqueries are performed in an increasing length order).*

Proof:

Let $q = q_1 q_2 \ldots$ be a partitioning of the query sequence q in increasing length order. Let the initial query radius be r, and let the dimensionality of the MR index structure be d. We compare two cases: 1) database is searched first for q_1 and then for q_2, and 2) database is searched first for q_2 and then for q_1.

Case 1: Let r_1 denote the distance of q_1 to an MBR. As shown in Lemma 1, $r_1 \approx r \cdot \sqrt{\frac{|q_1|}{|q|}}$. Let ϵ_1 be the refined radius after q_1. Using the radius refinement formula in Lemma 3 we have

$$\begin{aligned}\epsilon_1 &= \sqrt{r^2 - r_1^2} \\ &= \sqrt{r^2 - r^2 \cdot \frac{|q_1|}{|q|}} \\ &= r \cdot \sqrt{1 - \frac{|q_1|}{|q|}} \\ &= r \cdot \sqrt{\frac{|q|-|q_1|}{|q|}}.\end{aligned}$$

As a result, the final implicit search radius for q_2 after the initial radius refinement by q_1 =

$$\begin{aligned}r_{f_1} &= \epsilon_1 \cdot \sqrt{\frac{|q|}{|q_2|}} \\ &= r \cdot \sqrt{\frac{|q|-|q_1|}{|q|}} \cdot \sqrt{\frac{|q|}{|q_2|}} \\ &= r \cdot \sqrt{\frac{|q|-|q_1|}{|q_2|}}.\end{aligned}$$

Case 2: As in case 1, it can be shown that the final implicit search radius is

$$r_{f_2} = r \cdot \sqrt{\frac{|q|-|q_2|}{|q_1|}}.$$

Next, we compare the implicit search radii from cases 1 and 2. Since $q = q_1 q_2 ...$ is a partitioning of the initial query, we know that

$$\begin{aligned} & |q_1| + |q_2| &&\leq |q| \\ \Rightarrow\ & (|q_1| + |q_2|) \cdot (|q_2| - |q_1|) &&\leq |q| \cdot (|q_2| - |q_1|) \\ \Rightarrow\ & |q_2|^2 - |q_1|^2 &&\leq |q| \cdot |q_2| - |q| \cdot |q_1| \\ \Rightarrow\ & |q| \cdot |q_1| - |q_1|^2 &&\leq |q| \cdot |q_2| - |q_2|^2 \\ \Rightarrow\ & |q_1| \cdot (|q| - |q_1|) &&\leq |q_2| \cdot (|q| - |q_2|) \\ \Rightarrow\ & \frac{|q|-|q_1|}{|q_2|} &&\leq \frac{|q|-|q_2|}{|q_1|} \\ \Rightarrow\ & r_{f_1} &&\leq r_{f_2}.\end{aligned}$$

This implies that the final implicit search radius is minimized if the subqueries are performed in an increasing length order. □

3.4.2 Expected number of subqueries

The refined search algorithm has additional CPU cost due to multiple in-memory subqueries. For example, let q be a query sequence of length 25. If a base-2 MR index structure is used, then q is partitioned into 3 subqueries of lengths 1, 8, and 16. This corresponds to the 1s in the base-2 representation of $|q|$, $25 = (11001)_2$. On the other hand, if a base-3 MR index structure is used, then q must be partitioned into 5 subqueries of lengths 1, 3, 3, 9, and 9. These partitions correspond to the base-3 representation of $|q|$, $25 = (221)_3$. The number of subqueries for a base-i MR index structure can be computed as the sum of the digits in base-i representation of q. In our example, $(11001)_2$ results in 1+1+0+0+1=3 subqueries and $(221)_3$ results in 2+2+1=5 subqueries. The additional CPU cost is minimized when the number of subqueries is minimized. Lemma 5 proves that the expected number of subqueries is minimized if a base-2 MR index structure (i.e. resolutions of powers of 2) is used. This lemma is analogous to Lemma 2 that proved the optimality of base-2 MR index structure for performing a single longest prefix search.

Lemma 5 *If queries whose lengths are uniformly distributed between 1 and N (for some large integer N) are performed on the base-i MR index structure with equal probability, for $i \geq 2$, then the expected number of subqueries is*

$$N_i = \frac{(i-1) \cdot log_i N}{2}$$

Proof:

Let N_i represent the average number of subqueries for base-i MR index structure. The sum of digits of the base-i representation of $|q|$ is equal to the number of subqueries on MR$_i$. Define $h = log_i N$. Let S be the set of h digit, base-i representations of all the numbers from 1 to N. The numbers in S have a total of $N \cdot h$ digits. Since numbers 0, 1, ..., $i-1$ occur in S with the same frequency, each of them occurs $\frac{N \cdot h}{i}$ times. We can find the average number of subqueries by dividing the sum of the digits in S by $|S|$. As a result, we have:

$$\begin{aligned} N_i &= \frac{1}{N} \cdot \sum_{k=0}^{i-1} k . \frac{N \cdot h}{i} \\ &= \frac{h}{i} \cdot \sum_{k=0}^{i-1} k \\ &= \frac{h \cdot (i-1)}{2} \\ &= \frac{(i-1) \cdot log_i N}{2} \qquad \square \end{aligned}$$

Lemma 5 implies that the number of subqueries is minimized when powers of 2 are used as resolutions of the MR index structure since $\frac{(i-1) \cdot log_i N}{2}$ decreases as i decreases.

3.5 Nearest neighbor queries

A *k-nearest neighbor (k-NN) query* seeks the k closest subsequences from the database to a query string s. We perform a k-NN query in two phases. In the first phase, the k closest MBRs in the index structure are determined by an in-memory search on the index structure using the maximal resolution in the MR index structure. Once the k closest MBRs are determined, the algorithm reads the subsequences contained in these MBRs, and finds the k^{th} smallest distance of these subsequences to the query sequence. We represent this distance by r_1. Note that, generally a small percentage of the database is processed at this stage. In the second phase, we perform a range query using r_1 as the query radius. Figure 5 presents the complete k-NN search algorithm. It is guaranteed that the k nearest neighbors are retrieved in the second phase. This is because, the distance to the actual k^{th} nearest neighbor is at most r_1.

Korn, Sidiropoulos, Faloutsos, Siegel, and Protopapas [14] propose a similar k-NN search. The authors propose a technique in which k closest

```
Algorithm k-NN-SEARCH($(q, k)$)

• First Phase:

    1. $\mathcal{B}$ := The set of $k$ closest MBRs to the query
       $q_1$, where $q_1$ is the longest prefix of $q$ whose
       resolution appears in the MR index structure.

    2. Read the sequences contained in the MBRs in
       $\mathcal{B}$.

    3. $r_1$ := the distance of $q$ to the $k^{th}$ closest se-
       quence in the MBRs in $\mathcal{B}$.

• Second Phase

    1. RANGE_SEARCH$(q, r_1)$

    2. Return the $k$ closest strings in the answer set.
```

Figure 5. k-nearest neighbor algorithm.

points are obtained in the first phase using an approximate distance function. The actual distance to these points is computed, and a range query with the greatest actual distance is performed in the second phase. Seidl and Kriegel [24] propose an optimal iterative k-nearest neighbor search technique. They iterate over both the feature and the object spaces to ensure that no unnecessary objects are accessed. Our algorithm is closer in spirit to the former algorithm except that, we work with MBRs instead of data points in the first phase.

3.6 Comparison of the methods

The MR index structure alleviates a number of problems of the Prefix Search and Multipiece Search techniques that arise from their fixed structure. These problems are mainly the large amount of redundant computation and redundant disk page reads.

The Multipiece Search technique can have as much as $|q_{max}|/w$ subqueries, where q_{max} is the longest possible query. On the other hand, the upper bound on the number of subqueries for our method is $log(|q_{max}|) - log(w)$, where w is the minimum window length. This means a dramatic reduction in the number of subqueries. The other advantage of our technique is that disk reads are performed only after the last (longest) subquery, as opposed to the Multipiece Search technique which performs disk reads after each subquery.

Each subquery of the MR index structure refines the radius of the query. Since the search volume is proportional to ϵ^{dim}, even small refinements in the radius can prevent large number of disk reads. For example, if 10 dimensions are used in the index, then a 5% decrease in the radius will decrease the search volume by 40%. Therefore, the total search volume, hence the amount of computation and disk reads, for our technique is much lower than the Prefix Search technique.

However, there is a drawback of the MR index structure in that it uses more space than both Prefix Search and Multipiece Search. This is because it stores index structures at various resolutions. Since all the preprocessing steps must be done in memory, the index structure must fit into memory. In Section 5, we will present several methods that shrink the size of the MR index structure without much of a drop in performance.

4 Experimental results

We carried out several experiments to compare different search techniques, and to test the impact of using different parameters on the quality of our method. We used three datasets in our experiments. The first dataset is composed of stock market data taken from *chart.yahoo.com*. This dataset contains information about 556 companies for 1024 days. The total size of this dataset is approximately 4.3 MB. The second dataset is formed synthetically by adding four sine signals of random frequencies and some amount of random noise. This dataset is composed of 500 time series data each of length 1024. The total size of this dataset is approximately 4.0 MB. The third dataset is the *high quality Australian Sign Language* (*AUSLAN*) dataset. This data consists of sample Australian Sign Language signs. 27 examples of each of 95 AUSLAN signs were captured from a native signer using high-quality position trackers and instrumented gloves. There are a total of 2565 signs of various lengths in this dataset. We chose the first 1024 entries of each time series in this dataset. The total size of this dataset is slightly more than 20 MB. This dataset is publicly available at *http://kdd.ics.uci.edu*.

We normalized the entries of each of the time sequences in both of these datasets by dividing all the entries of each time sequence to the maximum of the absolute values of all the entries of that time sequence. We considered variable length queries of lengths between 16 and 1024. We assumed that the lengths of the queries are uniformly distributed. Each query sequence is generated by averaging two randomly selected subsequences from the database. The radii for these queries are selected uniformly between 0.10 and 0.20. The experimental results are computed as the average of 100 such queries. We assumed that the index structure fits into memory and that the page size was 8K for these experiments.

In order to obtain data at various resolutions, we had to transform the data to compact the time sequences and obtain MBRs of appropriate dimensionality. We experimented with three transformations, namely DFT, Haar wavelet [21, 26] and DB2 (Daubechies) wavelet. We also run some of the experiments using other wavelet bases such as DB3, DB5, SYM2 (Symlet), SYM3, SYM5 and several Coeiflets. However, we did not get any improvements with these basis functions. Haar wavelet performed slightly better than DFT. DB2 performed poor for lower dimensions, but its performance was comparable to Haar for higher dimensions.

Note that one can consider using other dimensionality reduction techniques like PAA [13] instead of DFT or wavelets. However, the experiments in [18] show that PAA is worse than DFT, Haar wavelets, and Daubechies wavelets. Our experiments with string data [8] also confirm that wavelets perform better than other techniques for our sliding window based index structure. There is no clear winner among different dimensionality reduction techniques. Therefore, we will report all our experiments using Haar wavelets.

Figure 6 shows the trails for IBM's closing prices for w = 16, when it is compressed to two dimensions using DFT, Haar, and DB2 wavelets. The figure shows that the trails are smooth for all of them. A smooth trail implies that the resulting subsequences can be indexed efficiently. We observed similar trails for other time series data as well.

4.1 Range queries

Our first set of experiments considered range queries. We measured precision and I/O for four different techniques: Sequential Scan, Prefix Search, Longest Prefix Search, and MR index. The results are presented in Figures 7 and 8. The results for Multipiece Search are not presented as it was not competitive with other methods: the total number of page accesses was even more than the total number of pages on disk.

We plot the precision and I/O (number of disk reads) as a function of the number of dimensions (coefficients) of the transformed data set. We define precision as the number of MBRs that contain

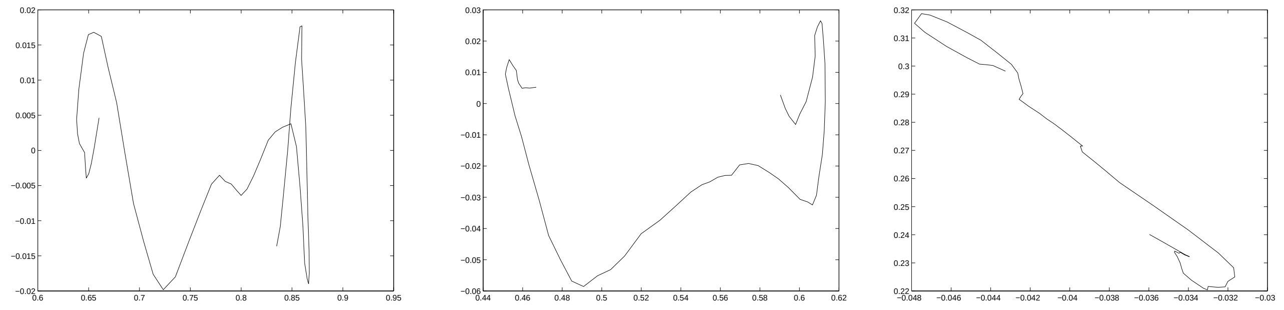


Figure 6. The trails of the IBM's stock for w = 16, when it is compressed to two dimensions using (a) DFT, (b) Haar, and (d) DB2.

result sequences divided by the number of MBRs in the candidate set. Note that for Sequential Scan, the candidate set is the entire database. The performance of Sequential Scan is not affected by the number of dimensions.

According to Figures 7(a) and 7(b), the MR index and LPS perform better than Prefix Search and Sequential Scan for all dimensionalities. For the stock market dataset, the precision of the MR index structure is more than five times better than Prefix Search and more than 15 times better than Sequential Scan. For the synthetic dataset, the precision of the MR index structure is three times better than Prefix Search and Sequential Scan. As compared to LPS, the MR index is about 15% better for the stock market dataset and 35% better for the synthetic dataset. This improvement is an indication of the performance gain due to iterative reduction in query range as we traverse down the rows of the MR index structure. The relative differences in the precision levels for the two datasets are due to the differences in their data distributions.

Figures 8(a) and 8(b) compare the I/O overhead of the four above mentioned techniques. For the stock market dataset, the number of page reads for the MR index structure is less than one-sixth of that for Prefix Search and less than one-seventh of that for Sequential Scan. For the synthetic dataset, the number of page reads for the MR index struc-

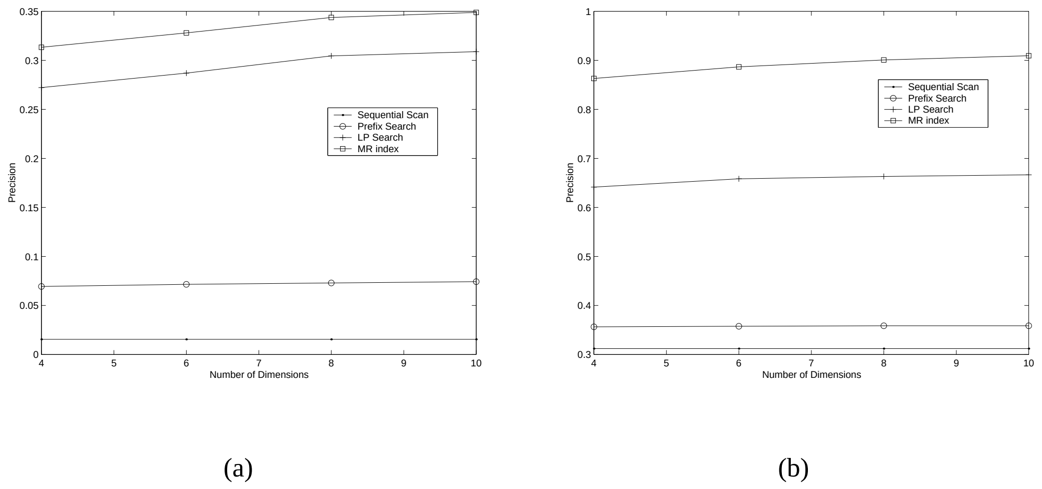


Figure 7. Precision for (a) stock market (b) synthetic data for different dimensionalities.

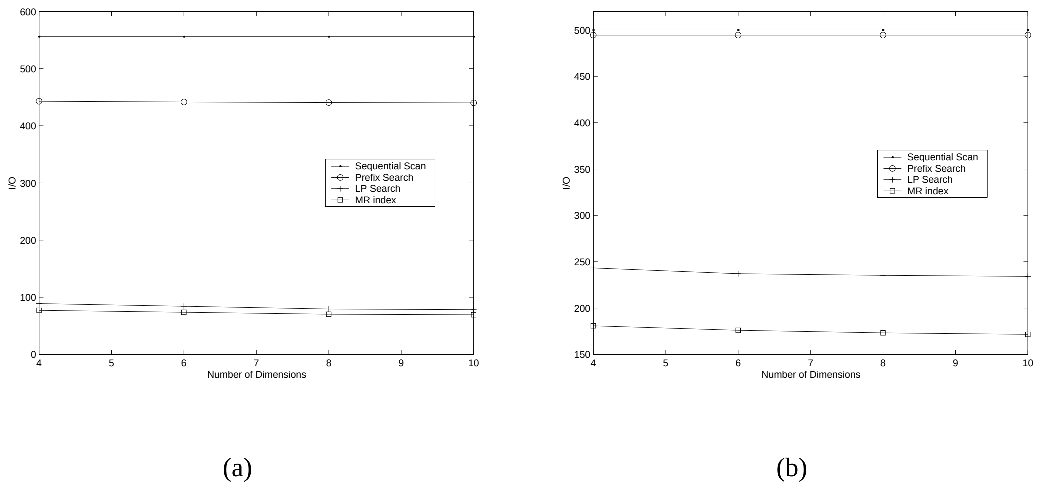


Figure 8. Number of disk reads for (a) stock market (b) synthetic data for different dimensionalities.

ture is less than one-third of that for both Prefix Search and Sequential Scan.

It should also be noted from the above four figures that the relative performance of the MR index structure improves as the number of dimensions increases. For example, by using 10 dimensions instead of 4 dimensions, the number of page accesses for the MR index structure decreases by 13%, while that for Prefix Search decreases by less than 1% for both datasets. Similarly, precision for the MR index structure increases by 13%, while that for Prefix Search increases by only 8% for both datasets.

As the number of dimensions increases, the number of disk I/Os drops slowly in Figures 8(a) and 8(b). This can be explained as follows. The

MR index structure stores only two vectors for each MBR, one for lower coordinates and one for higher coordinates, in order to decrease the size of the index structure. This is because storing the points in the MBR increases the size of the index structure dramatically. When the index structure becomes too large to fit into memory, the search technique incurs disk I/Os even in the index search phase. Since the MR index structure does not store the individual points, it postprocesses all the candidate MBRs. Hence, although the number of dimensions increases, there is still an imprecision due to the use of MBRs for all the points within that MBR.

Another important observation is that Prefix Search reads almost all the pages. The expected length of a query is 512. Since the minimum query length is 16, the expected implicit radius for a query is $\sqrt{512/16}$ (approximately 5.6) times larger than the actual range. This means that for ϵ between 0.1 and 0.2, the Prefix Search technique examines almost the entire search space.

The subqueries in the MR index structure correspond to binary representation of the ratio of the length of the query sequence to the minimum window size. Therefore, the expected number of subqueries for the MR index structure is $(log(1024) - log(16))/2 = 3$. In our experiments, the average number of subqueries was 3.08, which is pretty close to the theoretical results. On the other hand, for Multipiece Search, the expected number of subqueries is $512/16 = 32$, more than 10 times higher.

4.2 Estimating the total cost

Although Figures 7 to 8 give us information about CPU cost (precision determines the candidate set size, which in turn determines the number of arithmetic operations), and the I/O cost (number of disk page reads), determining the overall cost requires an estimation of the relative cost of arithmetic operations to disk page accesses. The cost of an arithmetic operation is usually much less than the cost of a disk page read. However, the cost of a disk page read itself can vary based on whether the read is sequential or random. In case of random I/O, there is a high seek and rotational latency cost for each page access. However, for sequential reads, disk head is at the start of the next page to be read, thus avoiding the seek cost. As a result, the cost of reading pages in a random order is much higher than the cost of reading them sequentially. We assume this ratio to be 12 [22, 23]. (There are also several other factors like buffering and prefetching that affect the page access cost. However, we do not consider these effects in our analysis.)

We normalize all costs to the number of sequential disk pages read. The total cost of a query is then computed as follows:

Total Cost = *CPU Cost* + *I/O Cost*, where

CPU Cost = *Candidate Set Size* $\cdot$ c, and

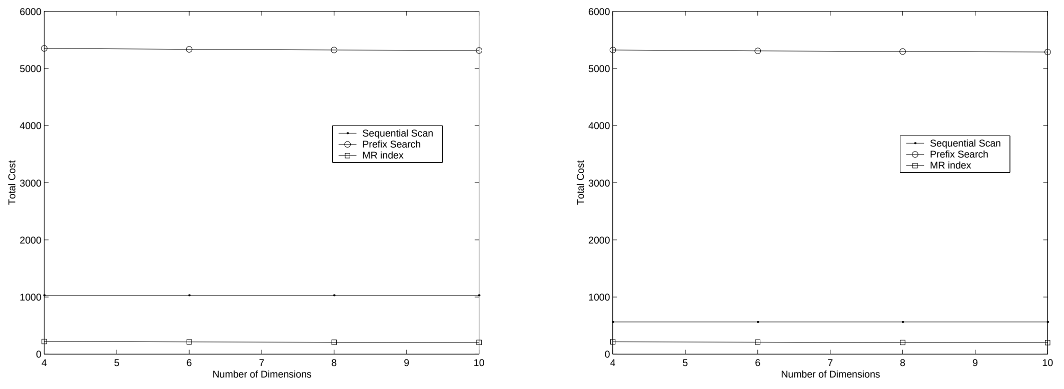
I/O Cost = k $\cdot$ *Number of page reads*

The constant c converts the cost of computing the distance between two sequences to the cost of a sequential disk page read. It was experimentally estimated to be between 1/300 and 1/600 depending on the hardware architecture.

The constant k depends on the search algorithm: if the pages are accessed sequentially then it is 1, if the pages are accessed in random then it is 12 [22, 23]. Prefix Search performs random page reads; as a result the corresponding constant is 12. In the MR index structure, a candidate set of database sequences is determined by accessing the MBRs corresponding to the last subquery. Disk accesses are then carried out to read this set of candidate sequences. Since the candidate sequences are accessed in the disk placement order, the average cost of a page read is much less than the cost of a random page read. We used the cost model proposed in [22] to find the cost of reading a subset of pages in sequential order. Typically, the value of k for the MR index structure lies between 2.0 and 3.0. We also validated this experimentally by reading subsets of pages in sequential order from a real disk.

The total cost comparisons corresponding to Figures 7 to 8 are presented in Figures 9(a) and 9(b). The results for the LPS technique are not presented here, because at this scale they become similar to the results of the MR index structure.

It can be seen from these figures that the MR index structure performs 2 to 4 times better than Sequential Scan, and more than 20 times better than Prefix Search for both datasets. Prefix Search always has the highest cost among the three techniques. This is because Prefix Search reads almost every disk page in a random order. As a result, the cost of disk reads dominates CPU time.



(a) (b)

Figure 9. Total cost of range queries for (a) stock market (b) synthetic data for different dimensionalities.

Although, Prefix Search is a good method for subsequence search when the query size is predefined, we conclude that it is not suitable when the query size is variable.

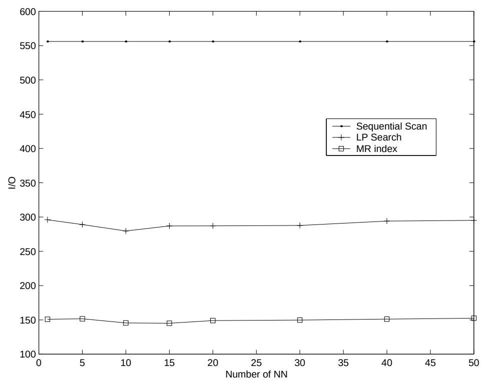
4.3 Nearest neighbor queries

Our second set of experiments considers arbitrary length k-NN queries on stock market and synthetic datasets for $k = \{1, 5, 10, 15, 20, 30, 40, 50\}$. Figures 10(a) and 10(b) present the number of disk reads for Sequential Scan, LPS technique, and the MR index structure. The results for the Prefix Search and Multipiece Search techniques are not presented, because they were not competitive.

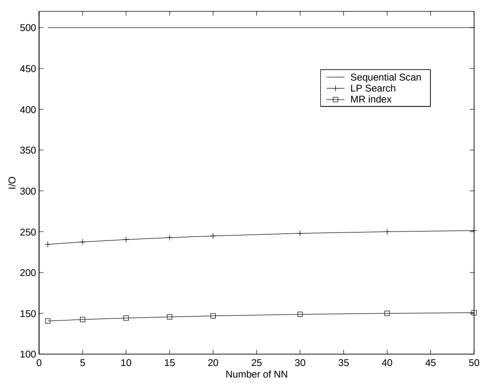
According to our experimental results, the MR index structure runs more than 3 times faster than sequential scan, and 2 times faster than the LPS technique for both datasets. The other point to note is that the number of disk reads does not increase significantly with the number of nearest neighbors.

4.4 Effect of query length

We inspect the effect of query length on the performance of the MR index structure, Prefix Search, and Sequential Scan. We used resolutions $w = \{16, 32, 64, 128, 256, 512\}$ for the MR index structure for this experiment and w = 16 for Prefix Search. The number of dimensions for this experiment is reduced to four using Haar wavelets. In this experiment we performed range queries for ϵ randomly selected between 0.1 and 0.2 for $|q| = \{16, 32, 64, 128, 256, 512\}$. The average of

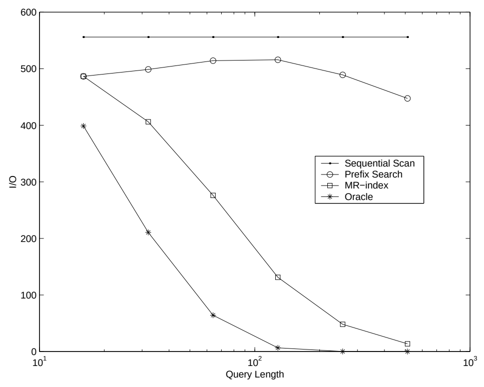


(a)

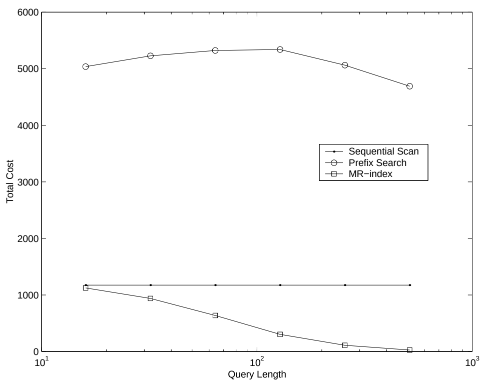


(b)

Figure 10. Number of disk reads for NN queries for (a) stock market (b) synthetic data.



(a)



(b)

Figure 11. (a) Number of disk reads (b) Total cost for range queries on stock market data for different query lengths.

100 queries is reported for each query length, .

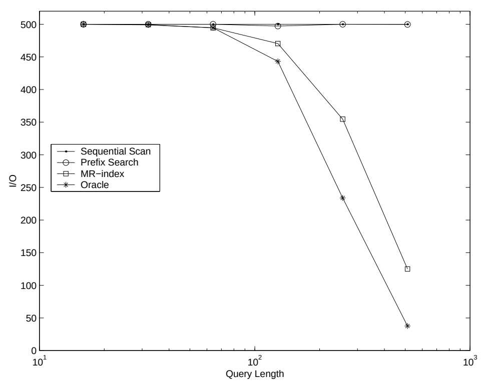
Figure 11(a) displays the number of disk I/Os performed for stock market dataset. In this figure, an additional curve, called *Oracle*, is plotted. We assume that Oracle knows which pages contain result subsequences in advance and accesses only those pages. The curve for Oracle shows the minimum number of disk I/Os required to answer the specified query. Prefix Search performs the same as the MR index structure for $|q|$ = 16. This is because the prefix of length 16 of a query is equal to the whole query when $|q|$ = 16. The MR index structure accesses smaller number of pages compared to both Sequential Scan and Prefix Search. The number of disk I/O for the MR index structure decreases as $|q|$ increases. This is because, the answer set gets smaller for large $|q|$, and the MR index structure can capture this since it uses all the information within $|q|$. Unlike other competing techniques, the number of disk I/O for the MR index structure changes in parallel with Oracle. This shows that the MR index structure scales well with respect to the query length. The gap between Oracle and the MR index structure corresponds to information lost due to dimensionality reduction and the use of an MBR to approximate a set of subsequences.

Figure 11(b) plots the total cost in terms of number of sequential page transfers for stock market dataset. The MR index structure always has the lowest cost and Prefix Search always has the highest cost. The performance of the MR index structure is similar to that of Sequential Scan for $|q| = 16$. This is because the result set is so large that almost all the disk pages contain at least one result sequence (see the graph for Oracle in Figure 11(a).).

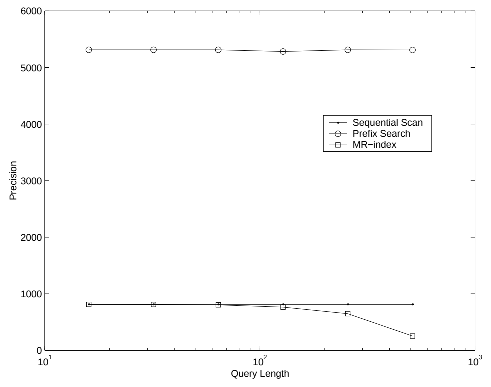
Figure 12(a) and Figure 12(b) show the number of disk I/Os performed and the total cost for the synthetic dataset. The MR index structure accesses the smallest number of disk pages. Prefix Search reads almost all the pages for this data set. The total cost of the MR index structure is less than both Prefix Search and Sequential Scan. For short queries, the performance of the MR index structure is similar to that of Sequential Scan. This is because almost all the disk pages contains at least one answer sequence (see the graph for Oracle in Figure 12(a).). As query length increases, the size of the answer set decreases. The decreasing cost graph for the MR index structure shows that the MR index structure prunes the search space much better than Prefix Search as query becomes longer.

4.5 Effect of database size

We used the *AUSLAN* dataset to test the effect database size on Sequential Scan, Prefix Search, and the MR index structure. In order to carry out this experiment, we split the *AUSLAN* dataset into nine equal sized datasets D_1, D_2, ..., D_9. Later, we formed datasets D'_1, D'_2, ..., D'_9 of various sizes by merging the smaller datasets cumulatively (i.e. $D'_1 = D_1$, and $D'_i = \cup_{j=1}^{i} D_j$, for $2 \leq i \leq 9$, where $\cup$ is the union operator.). We performed range queries of random lengths between 16 and 1024 on these datasets. The query radius



(a)



(b)

Figure 12. (a) Number of disk reads (b) Total cost for range queries on synthetic data for different query lengths.

is also selected randomly between 0.1 and 0.2.

Figure 13 displays the total cost for Sequential Scan, Prefix Search, and the MR index structure. The MR index structure always has the lowest cost. The total cost of the MR index structure increases linearly with the database size as well as other discussed techniques. This is because of the grid structure of the MR index: the MBRs at different columns of the MR index structure are pruned independently. Increasing the size of the database increases the number of columns. Since the probability of pruning a column is independent of the number of columns the total cost of the MR index structure increases linearly with the number of columns. We conclude that the MR index structure scales well with respect to database size.

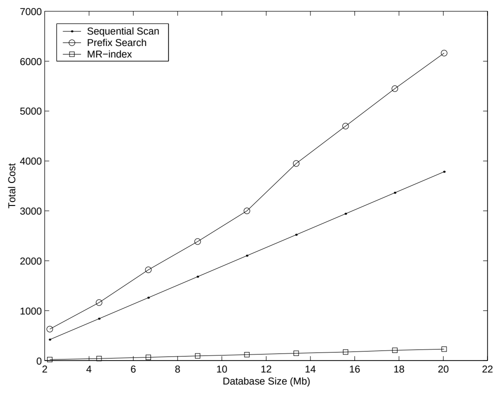


Figure 13. Total cost of range queries in terms of number of sequential page transfers for different database sizes.

5 Index compression

The MR index structure performs better than all the other techniques discussed in this paper in terms of precision, number of page accesses, and total cost. However, the method keeps information about the data at more than one resolution in the index structure. Therefore, the MR index structure uses more space than other index structures like the *I-adaptive* index that maintain information at a single granularity level. In order to perform the preprocessing steps of the search in memory, the index structure must fit into memory. If the index structure does not fit into memory, then the method will incur disk page reads for accessing the index. Therefore, the performance of the MR index structure could degrade if enough memory is not available. In this section, we consider index compression techniques with which memory requirements of the MR index structure can be made similar to the other index structures.

5.1 Compression methods

We experimented with two index compression methods for the MR index to shrink the size of the index: *Fixed Compression* and *Greedy Compression*. These methods merge a set of MBRs to create an extended MBR (EMBR) of the same dimensionality that tightly covers and replaces the constituent MBRs. The MR index assumes that an index node contains subsequences that start at consecutive offsets. In order to preserve this property, only consecutive MBRs can be merged into an EMBR. Each merge operation increases the total volume covered by the index structure. As the total volume increases, the probability that a range query accesses one or more of these MBRs increases. This increases the size of the candidate sets. Therefore, we would like to merge as many MBRs as possible into an EMBR, while minimizing the increase in volume. We consider two strategies for merging MBRs. Let r be the desired compression rate.

1) *Fixed Compression* is based on the observation that in real applications consecutive subsequences are similar. As a result, this method merges the first r MBRs into an EMBR, second r MBRs into another EMBR, and so on. This is a very fast method. However, if consecutive MBRs are not similar, the performance drops quickly.

2) *Greedy Compression* tries to minimize the increase in the total volume occupied by the MBRs at each merge operation. The algorithm chooses the two consecutive index nodes that lead to a minimal volume increase at each step until the number of index nodes is decreased by the given

ratio r. This method adapts to the layout of the index nodes at each merge operation. However, since the two MBRs to be merged are determined independently at each step, some of the EMBRs can contain many more subsequences than others, leading to an unbalanced partitioning. This has the drawback that an access to one of the heavily populated index nodes will incur the overhead of reading large number of candidate subsequences. This decreases the precision of the index.

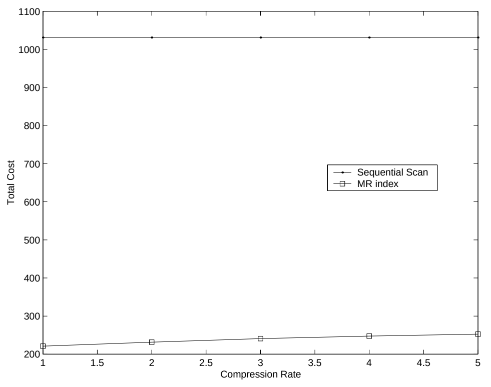
5.2 Experimental results

We ran several experiments using range queries to test the performance of the compression heuristics on the stock market dataset. In these experiments, we varied the compression rate from 1 (implying no compression) to 5 (maximum compression). The results are shown in Figure 14. As evident from the figure, the Greedy heuristic has a lower cost than the Fixed Compression technique for all compression rates. We obtained similar results for other datasets as well.

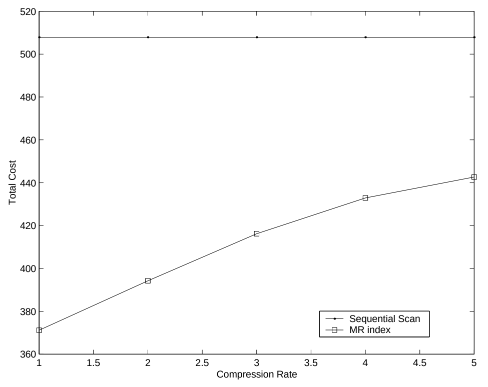
Figures 15(a) and 15(b) compare the total cost of range queries for Sequential Scan and MR index with Greedy Compression. We did not plot the results for Prefix Search since it performed worse than sequential results. When the compression rate is 5, the total cost of the MR index structure is still 4 times better than the cost of Sequential Scan, and more than 15 times better than that of Prefix Search for the stock market dataset. The total cost of the MR index structure is 20$\%$ less than that of Sequential Scan for the synthetic data set when the compression rate is 5.

In Figures 9(a) and 9(b), we already showed that Sequential Scan is better than Prefix Search when the MBRs of the I-adaptive index are not compressed. Since, the compressed MR index structure has lower cost than Sequential Scan, we conclude that the compressed MR index structure is also better than Prefix Search of equal size index structure.

Compression has another advantage. It increases the number of rows of the MR index structure, and as a result makes the index structure efficiently applicable to longer query sequences. For example, compressing the MR index to one-fifth, the number of rows of the MR index can be increased 5 times while keeping the storage cost invariant.



(a)



(b)

Figure 15. Total cost of range queries for (a) stock market (b) synthetic data for different compression rates.

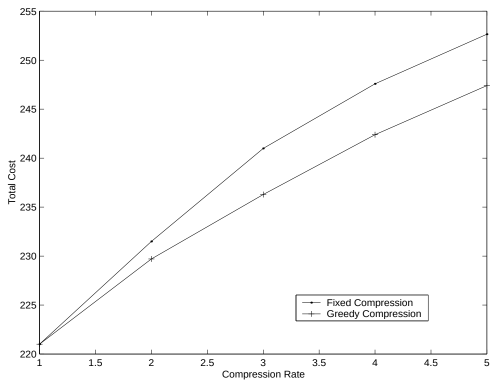


Figure 14. Total cost of range queries for different index compression techniques on stock market data.

6 Discussion

In this paper, we considered the problem of variable length queries for subsequence matching for time sequences. This is the most general version of the Euclidean distance based time series similarity problem since no restriction is imposed on the length of the sequences. We proposed a new indexing method called MR index, that stores information about the data at different resolutions. This index structure overcomes the limitations of the current search techniques. We proved the superiority of this multi-resolution index structure over single-resolution index structures.

We first proposed a naive range search technique called Longest Prefix Search. This technique performs a prefix search on the largest possible resolution available in the in index structure. Based on this technique, we proved that using powers of two in the index resolutions maximizes the accuracy of the MR index structure.

Later, we presented improved algorithms for both range queries and nearest neighbor queries. The range query algorithm splits the given query sequence of arbitrary length into several subqueries corresponding to the resolutions available in the index structure. The ranges of the subqueries are successively refined, and only the last subquery performs disk reads. We proved that a top-down search of the index structure maximizes the efficiency of the search technique. As a result of these optimizations, the MR index performed 4 to 20 times better than the other methods including Sequential Scan for range queries.

The k-nearest neighbor algorithm runs in two phases. In the first phase, the distance of the query to the MBRs are computed, and the distance to the k^{th} closest subsequence in the k closest MBRs is used as the radius for a range query in the second phase. This algorithm performed three times better than Sequential Scan on our datasets.

Since the MR index stores information at multiple resolutions, the size of the MR index structure is larger than index structures based on a single resolution. We proposed two methods to compress the index structure without losing much information: Fixed Compression and Greedy Compression. These methods compress the index structure by merging some of the MBRs. Greedy Compression has better results since it minimizes the information loss (increase in volume). The MR index performed 3 to 15 times better than other techniques even after compressing the index to one-fifth.

The simple grid structure of our index provides a good opportunity for parallel implementation. The index structure can be partitioned into subsets of columns and each subset can be mapped to a separate processor with its own disk. No communication would be needed among the processors, except to merge the results at the end.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FODO*, Evanston, Illinois, October 1993.

[2] R. Agrawal, K. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, Zürich, Switzerland, September 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, Atlantic City, NJ, 1990.

[4] K.-P. Chan and A.W.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, Sydney, Australia, March 1999.

[5] K.K.W. Chu and M.H. Wong. Fast time-series searching with scaling and shifting. In *PODS*, Philadelphia, PA, 1999.

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

[7] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[8] T. Kahveci and A. Singh. An efficient index structure for string databases. In *VLDB*, pages 351–360, Roma, Italy, September 2001.

[9] T. Kahveci and A. Singh. Variable length queries for time series data. In *ICDE*, Heidelberg, Germany, 2001.

[10] T. Kahveci, A. K. Singh, and A. Gürel. Similarity searching for multi-attribute sequences. In *SSDBM*, 2002.

[11] K.V.R. Kanth, D. Agrawal, and A. Singh. Dimensionality-reduction for similarity searching in dynamic databases. In *SIGMOD*, Seattle, WA, June 1998.

[12] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2001.

[13] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems Journal*, 2000.

[14] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical databases. In *VLDB*, pages 215–226, India, 1996.

[15] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity search for multidimensional data sequences. In *ICDE*, San Diego, CA, 2000.

[16] S. Park, W.W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *ICDE*, San Diego, CA, February 2000.

[17] C.-S. Perng, H. Wang, S.R. Zhang, and D.S. Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *ICDE*, San Diego, USA, 2000.

[18] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *ICDE*, pages 212–221, San Jose, CA, 2002.

[19] D. Rafiei and A.O. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, pages 13–25, Tucson, AZ, 1997.

[20] D. Rafiei and A.O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *FODO*, Kobe, Japan, 1998.

[21] R.M. Rao and A.S. Bopardikar. *Wavelet Transforms Introduction to Theory and Applications*. Addison Wesley, 1998.

[22] B. Seeger. An analysis of schedules for performing multi-page requests. *Information Systems*, 21(5):387–407, 1996.

[23] B. Seeger, P.-A. Larson, and R. McFayden. Reading a set of disk pages. In *VLDB*, pages 592–603, Dublin, Ireland, August 1993.

[24] T. Seidl and H.P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, 1998.

[25] C. Shahabi, X. Tian, and W. Zhao. TSA-tree: A wavelet-based approach to improve the efficieny of multi-level surprise and trend queries. In *SSDBM*, 2000.

[26] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[27] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, San Jose, CA, 2002.

[28] C. Wang and X. S. Wang. Supporting content-based searches on time series via approximation. In *SSDBM*, 2000.

[29] Y.-L. Wu, D. Agrawal, and A. El-Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *CIKM*, pages 414–421, 2000.

Tamer Kahveci recieved the BS degree in Computer Engineering from Bilkent University, Turkey in 1999. He is currently a Ph.D candidate in Computer Science at University of California at Santa Barbara. His main research interests include databases and bioinformatics.

Ambuj K. Singh is a Professor in the Department of Computer Science at the University of California at Santa Barbara. He received his B.Tech.(Hons) in Computer Science and Engineering from Indian Institute of Technology, Kharagpur in 1982, his Ph.D. in Computer Science from the University of Texas at Austin in 1989. His research interests are in the areas of bioinformatics, distributed systems, and databases.