Chapter 4

Grid-Based Indexing for Large Time Series Databases

In this chapter, we extend the idea of datawise dimensionality reduction for indexing large time series datasets.

4.1 Introduction

Examples of time series data includes stock prices, weather data, exchange rates, medical information, *et al.* Each time series is a sequence of real numbers, and each element of which represents the value at a point in time. Similarity search in time series is useful in its own right as a tool for exploratory data analysis. The problem, therefore, has attracted the attention of many researchers. However, description of time series requires an enormous amount of real numbers, real world datasets are typically at the gigabyte or terabyte level.

The most promising similarity search techniques perform dimensionality reduction followed by the use of a SAM (Spatial Access Method) such as R-tree [23] and its many variants[7, 26] in the transformed space. These techniques include Discrete Fourier Transform (DFT) [3], Piecewise Aggregate Approximation (PAA)[29, 39] and Adaptive Piecewise Constant Approximation (APCA)[27]. They map each Fourier coefficient, aggregate segment and adaptive aggregate segment onto one dimension of an index tree. The efficiency of the above-mentioned indexes depends on the fidelity of the approximation in the reduced dimensionality space. The more accurate the approximation gets (i.e. the more coefficients retained), the higher the dimensionality of the transformed space becomes. Unfortunately, recent work by Chakrabarti and Methrotra, and others suggests that the R-tree and its variant multidimensional greater than $8 \sim 12$. To overcome this difficulty, known as the *dimensionality curse*, a compressed indexing structure VA-file[38] has been proposed. Scanning a compressed index file generates a small set of candidates. The exact answer is obtained through refinement checking to this small candidate set, which needs fewer random accesses to data files. By taking into account that real datasets are always skewed, new versions of VA-file such as CVA-file[4], VDR-file [5] and C^2VA[16] have been proposed. In these improvements, the techniques of dimensionality reduction and file compression are efficiently integrated.

The main contributions of DDR are the introduction of a simple, but highly effective compression technique, called grid-based Datawise Dimensionality Reduction (DDR) and the introduction of a technique which can index this representation. The basic idea of DDR

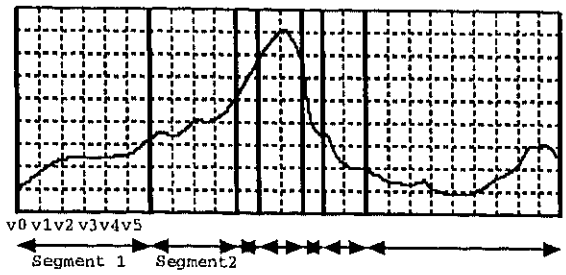

Figure 4.1: Subsequence which has near values v_i is represented one segment.

is illustrated in Figure 4.1. In the figure, a time series data represented by a series of value $v_1, v_2, ...$, is plotted. Our approach is to approximate the value by quantization, then group "near" values together and represent them with a single approximate value. Each group is called a *segment*, which will be defined precisely in the next section. In the figure, using a certain threshold, v_0 through v_5 fall in a same rectangle so they are group together and are represented by one value: the quantization of v_0. Grouping 6 points together reduces the size to 1/6 of the original data, and quantization reduces the size further.

The rest of this chapter is organized as follows. A review of related work on the similarity search in time series immediately follows. In Section 4.2 we introduce necessary definitions and notation as well as the overview of our approach. In Section 4.3 we introduce the DDR representation and the two distance measures defined on it. Section 4.4 discusses the formal description of the construction and indexing of DDR representation. Section 5.4 contains a comprehensive experimental comparison of DDR with some competing techniques. Section 4.6 offers some conclusions.

4.1.1 Related Work

Besides the work mentioned above, recent work by Keogh and Yi is closely related to this research.

DFT [3] based on the well known Fourier Transform(FT) is an efficient dimensionality reduction. However, as the length of time series becomes longer, much more coefficients are necessary to approximate the data, which increases the storage requirements.

In [39], Yi *et al.* proposed an approximation of a time series by dividing it into equal-length segments. Each segment is represented by the mean value of the data points that fall within it, and all the mean values together represent the original series. This representation is called Piece-wise Aggregate Approximation (PAA).

Keogh *et al.* generalized PAA to APCA (Adaptive Piecewise Constant Approximation) which allows a segment to have arbitrary length. They use a Wavelet-based heuristic approach for finding the piecewise polynomial representation of the time series. To describe each segment, two numbers are used.

As shown in Figure 4.2, the main difference between DDR and other works is that DDR introduces the concept of *grid* which has 2-dimensional lattice structure. The horizontal axis corresponds to the time dimension and is partitioned into varying sized ranges. The vertical axis represents the quantized value of the data point. Grid-based representation takes into consideration the semantics of time series, hence enhances dimensionality reduction while keeping the accuracy of representation. Moreover, it is easy to construct the index structure using existing quantization approach such as CVA-file.

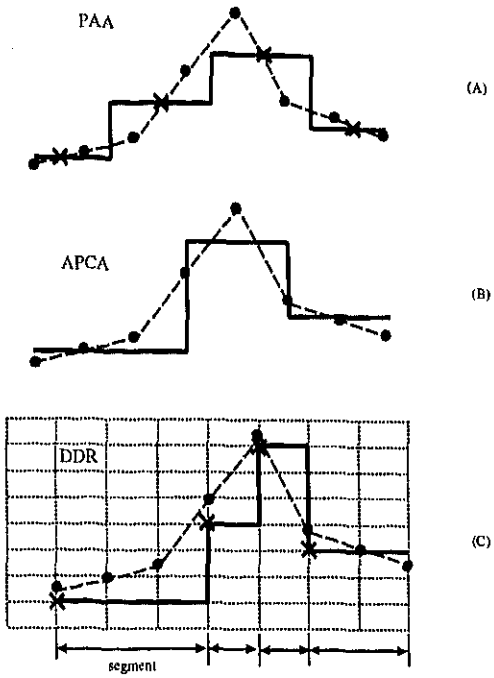

Figure 4.2: A comparison of the reconstruction of PAA, APCA and DDR.

4.2 Approach

4.2.1 Notation and Terminology

We begin by summarizing the symbols and terms. The data set of time series is $\mathcal{O}$, and an object $v \in \mathcal{O}$ is a time series which is represented by a sequence of values $v_1, v_2, \ldots, v_n$. All objects in $\mathcal{O}$ have the same length n, which is also referred to as *dimensionality*. Each v_i of v is called a *data point*.

The main problem we are to address is to find k-NN of a query q from $\mathcal{O}$. This also implies that the length or dimensionality of q is also n.

There are two types of similarity search for time series. One is *whole matching* and the other is *subsequence matching*. The former is to compare the whole sequences of the same

length n, while the latter tries to find a part of the sequence which matches the query. In this, we concentrate on whole matching.

The similarity between two time series is typically measured with Minkowski distance function (also known as L_p). More precisely, given two time series $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, the distance between u and v is usually defined by:

$$D(u, v) = \sqrt[p]{\sum_{i=1}^{n} (u_i - v_i)^p} \tag{4.1}$$

Figure 4.4 illustrates the definition of the distance function for time series data.

In order o find similar series, it is important to avoid the being mislead by the possibility that two time series may be similar, but at different scales, or at different offsets. For instance, a series of data points distribute on a curve of function $sin(x) + 1$ must be similar to the series on $sin(x)$. However, by the distance function defined above, the former series is more similar to a series of data points all having value 1. A simple way is to *normalize* the original series, that is, to map the data points to the range [0, 1). After normalization the series on $sin(x) + 1$ match $sin(x)$ completely, or, the distance between them becomes zero.

4.2.2 Overview of our Approach

Our approach is based on the concept of *grids*. The horizontal axis of a time series data is divided into divisions where each data point is located. The vertical axis is also divided for representing values of data points by quantization. Consequently, grids are obtained as

illustrated in Figure 4.2(C).

Since realistic data are always high dimensional, there will be too many grids to treat efficiently. Reduction of dimensionality helps to reduce the number of grids, and our approach to do so is by introducing *segment*, which merges certain adjoining grids together.

We assume that values of realistic time series data tend to change gradually as shown in Figure 4.3 (B), and rapidly changing patterns like the one shown in Figure 4.3 (A) are rare. This seems true by simple visual inspection for most real world datasets[27]. In other words, for a time series $v = (v_1, v_2, \ldots, v_n)$, there are many data points satisfying $v_i \doteq v_{i+1} \doteq \cdots \doteq v_{i+m}$. Consecutive data points satisfying such condition is grouped together into a *segment*, then the time series data can be represented as an array of segments. Note that the length of segments (the number of data points grouped together) are not fixed.

Within a segment, all values of data points can be mapped into a representative value with some error tolerance. By this way, we can represent each segment by a single representative value. This technique enables us to reduce the dimensionality of time series data.

Furthermore, instead of the values of data points themselves, we construct index files on their quantized approximation. Quantization as well as the grouping technique of segments contribute to drastic reduction of the size of index files.

The search procedure in our approach is performed in two phases. In phase 1 (*filtering* phase), the index file is scanned. The access method in this phase is sequential access which is much more efficient than random access. After the filtering of this phase, a smaller set of

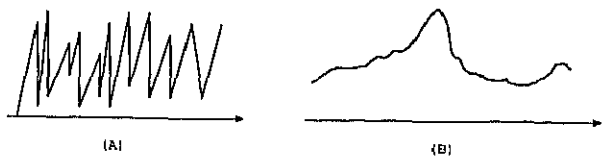

Figure 4.3: The realistic time series: (A) very rare case. (B) common case. (e.g., stock price of one day, or medical diagram.)

candidates is obtained. Then in the following phase 2 (*refinement* phase), a few pages are accessed to calculate exact distance from the query point q. It is highly desirable that the number of page accesses in phase 2 be as small as possible.

4.3 Data Representation

In this section we discuss how to determine grids. First, the normalized values are quantized, then the lengths of segments are determined. To guarantee the admissibility when DDR is implemented as an index, upper and lower bounds are also considered.

4.3.1 The DDR representation

We illustrate the representation of a given time series $(v_1, v_2, \ldots, v_n)$ by using an example. For clarity of presentation, the explanation is separated to two steps. In fact, as will be shown in Algorithm 3, the index file is constructed more efficiently rather than in two steps separately.

As the example, suppose the time series is given as below.

$$(v_1, v_2, \ldots, v_8) = (0.18, 0.24, 0.30, 0.62, 0.9, 0.45, 0.38, 0.32)$$

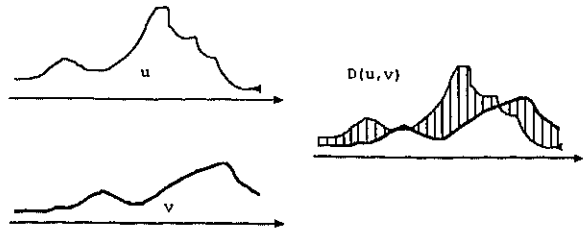

Figure 4.4: The distance function is the p-th root of the sum of the lengths of the vertical lines to the p-th power.

this series is approximated in the following way.

1. **quantization:** The time series is quantized to 2^b intervals in the vertical axe, where b is the number of bits used for approximating v_i. In the example, if we assumed that $b = 3$, then by $\alpha_i = \lfloor v_i \times 2^b \rfloor$, the quantized time series of the example becomes

$$(\alpha_1, \alpha_2, \ldots, \alpha_8) = ((001)_2, (001)_2, (010)_2, (100)_2, (111)_2, (011)_2, (011)_2, (010)_2)$$

2. **reduction:** If a value of a data point is close to that of its previous data point, then it is omitted. We use a tolerance parameter ϵ. Whether a data point is omitted or not is determined by Equation 4.2.

$$\alpha_{i+1} = \begin{cases} \text{omitted}, & -\epsilon \leq v_{i+1} - \alpha'_i \times h \leq h + \epsilon \\ \lfloor v_{i+1} \times 2^b \rfloor, & otherwise \end{cases} \qquad (4.2)$$

$$(i = 1, 2, \ldots, n)$$

In the equation, h is the height of grids ($= 1/2^b$) and α'_i is the representative value

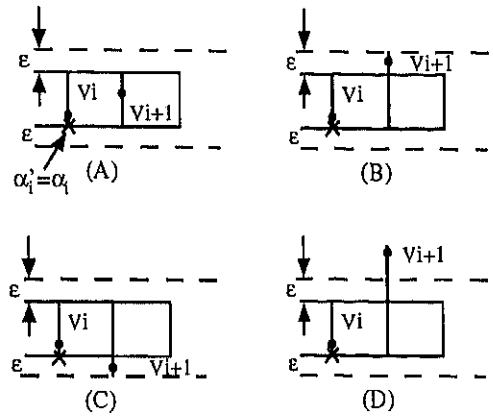

Figure 4.5: Determination of Segments. (A) Since v_{i+1} has the same quantized value as v_i, it is omitted (grouped in the same segment as v_i). (B) Although quantized value of v_{i+1} is greater than v_i, it is also omitted because it is still in the error tolerance with respect to v_i. (C) This case is similar to the case (B). (D) v_{i+1} is out of the error tolerance, and thus begins a new segment.

of the segment in which the quantized value α_i' belongs. The representative value of each segments is the quantized value of the first data point in the segment.

Figure 4.5 shows four cases for determining segments of quantized time series. For the illustrative convenience, we assume that the previous quantized value α_i is not omitted.

To illustrate how to reduce the dimensions, the range of error tolerance is added as shown in Figure 4.6. The data points in a gray rectangle belong to a single segment, so their values are omitted except the first one. In this example, ϵ is set to $h/2 = 1/16$, and after the reduction, the time series becomes

$$(\alpha_1, \alpha_2, \ldots, \alpha_8) = ((010)_2, \times, \times, (100)_2, (111)_2, (011)_2, \times, \times),$$

where "$\times$" denotes the omission of the data points. By Equation 4.2, v_1 is out of range $[-\epsilon, \epsilon + h] = [-1/16, 3/16]$ (α_0 is initialized to 0), so α_1 must be stored. α_2 is omitted

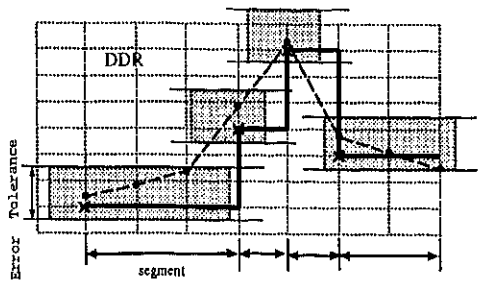

Figure 4.6: Reduction of data points of v with DDR technique. The four gray rectangles represent the ranges of error tolerance.

because $v_2 - \alpha_1 \times h = 0.24 - 0.125$ is in the range $[-1/16, 3/16]$. To determine whether v_3 can be omitted, α_1 is used because v_2 has been omitted. $v_3 - \alpha_1 \times h = 0.3 - 0.125 = 0.1775$ is also in the above range, thus α_3 is also omitted. Similarly, $v_4 - \alpha_1 \times h = 0.62 - 0.125$ exceeds the range so α_4 must be stored. The rest is calculated in the similar way.

There are two advantages of representing DDR based on the grid partition. First, data are stored in an index file effectively; second, unlike APCA technique, the omitted data point is easily determined.

Because the segments are variable-length, it is not known how many and which of the data points have been omitted. To record this information, an n-bit length bit pattern β is associated to the approximation of a time series data. If α_i is omitted, the corresponding bit β_i is set to 0. Otherwise it is set to 1. The structure of an entry representing one time series data is shown in Figure 4.7. In the case of the example above, the entry representing the series data becomes

$$(\overbrace{10011100}^{\text{Omission info.}}, \overbrace{010, 100, 111, 011}^{\text{quantized values}})_2$$

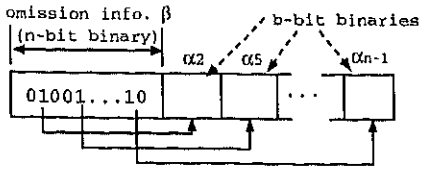

Figure 4.7: Representation of one time series. n-bit binary denotes which data points in v is omitted or not. The following boxes indicate quantized value (α_i) of data points (v_i)

4.3.2 Distance Measures Defined for DDR

Consider a time series v, which we convert to DDR representation α, and a query time series q. Instead of computing the exact distance $D(q, v)$ between q and v, we want to estimate the bound between q and α to exclude as many irrelevant series as possible. In index files, since the quantized values of data points instead of exact values are stored, we have to define distance measures between q and α which approximates $D(q, v)$. The function has to avoid false dismissals, while maximizing the filtering effect. Functions $D_{LB}(q, v)$ and $D_{UB}(q, v)$ are defined so that they are lower and upper bounds of D(q,u). These functions are used in the filtering phase for selecting candidates.

$D_{LB}(q, v)$ is defined as:

$$D_{LB}(\mathrm{q}, v) = \sqrt[p]{\sum_{i=1}^{n} (d_{LB}(q_i, v_i))^p} \tag{4.3}$$

where $d_{LB}(q_i, v_i)$ is lower bound value of two corresponding data points in q and v.

Similarly, $D_{UB}(q, v)$ is defined as:

$$D_{UB}(\mathrm{q}, v) = \sqrt[p]{\sum_{i=1}^{n} (d_{UB}(q_i, v_i))^p} \tag{4.4}$$

where $d_{UB}(q_i, v_i)$ is upper bound value of two corresponding data points in q and v.

To estimate the values of lower and upper bound, two cases are considered.

1. α_i **is stored.** The quantized value of v_i is available, because it is stored in an index file, as shown in Figure 4.8, $d_{LB}(q_i, v_i)$ and $d_{UB}(q_i, v_i)$ can be calculated by

$$d_{LB}(q_i, v_i) = \begin{cases} q_i - (\alpha_i + 1)h & q_i > (\alpha_i + 1)h \\ 0, & \alpha_i h \leq q_i \leq (\alpha_i + 1)h \\ \alpha_i h - q_i, & q_i < \alpha_i h \end{cases} \tag{4.5}$$

$$d_{UB}(q_i, v_i) = \begin{cases} q_i - \alpha_i h & q_i > (\alpha_i + 1)h \\ max((\alpha_i + 1)h - q_i, q_i - \alpha_i h) & \alpha_i h \leq q_i \leq (\alpha_i + 1)h \\ (\alpha_i + 1)h - q_i, & q_i < \alpha_i h \end{cases} \tag{4.6}$$

2. α_i **is omitted.** The value of v_i is estimated by α_i', the representative value of the segment to which v_i belongs. As mentioned above, representative value of a segment is the quantized value of the first data point in it. As shown in Figure 4.9, $d_{LB}(q_i, v_i)$ and $d_{UB}(q_i, v_i)$ are defined as

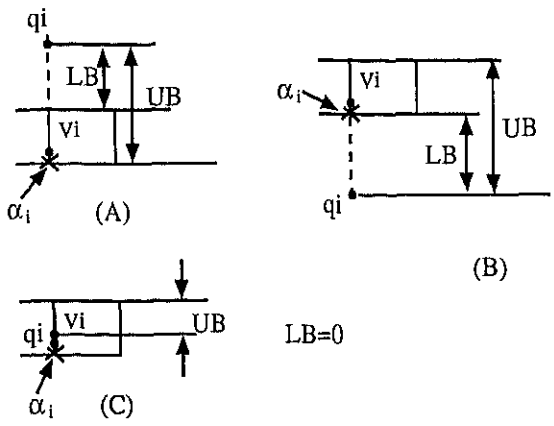

Figure 4.8: Lower bound and upper bound. (A) q_i is larger than upper value of quantized value α_i of v_i. (B) q_i is smaller than lower value of quantized value α_i of v_i. (C) q_i has the same quantized value as α_i of v_i.

$$d_{LB}(q_i, v_i) = \begin{cases} q_i - (\alpha_i' + 1)h - \epsilon & q_i > (\alpha_i' + 1)h + \epsilon \\ 0, & \alpha_i' h - \epsilon \leq q_i \leq (\alpha_i' + 1)h + \epsilon \\ \alpha_i' h - q_i - \epsilon, & q_i < \alpha_i' h - \epsilon \end{cases} \tag{4.7}$$

$$d_{UB}(q_i, v_i) = \begin{cases} q_i - \alpha_{i-1} h + \epsilon & q_i > (\alpha_i' + 1)h + \epsilon \\ max((\alpha_i' + 1)h + \epsilon - q_i, q_i + \epsilon - \alpha_i' h), & \alpha_i' h - \epsilon \leq q_i \leq (\alpha_i' + 1)h + \epsilon \\ (\alpha_i' + 1)h - q_i + \epsilon, & q_i < \alpha_i' h - \epsilon \end{cases} \tag{4.8}$$

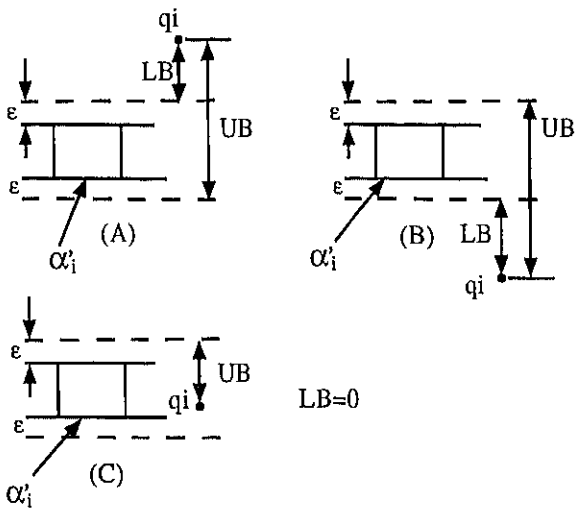

Figure 4.9: Lower bound and upper bound. (A) q_i is larger than upper value of quantized value α_i' of v_i with addition of error tolerance ϵ. (B) q_i is smaller than lower value of quantized value α_i' of v_i with subtraction of error tolerance ϵ. (C) q_i has the same quantized value as α_i' of v_i in error tolerance.

4.4 Indexing DDR

In this section, we formally describe the algorithms using in the construction and search of DDR. Based on the analysis in Section 4.3, the procedure of the construction is given in Algorithm 3. As shown in Figure 4.7, DDR has a flat structure. It is a sequence whose entries correspond to the original series data one by one. Each entry contains a header of n-bit, indicating the presence or omission of quantized values by setting or unsetting the corresponding bit, respectively.

It is easy to see that the complexity of the algorithm is $O(|\mathcal{O}|)$, the size of the constructed index file is $(1 + \tau b)n|\mathcal{O}|$ bits, where τ is the average reduction rate. The size of the original data file is $fn|\mathcal{O}|$ bits, where f is the number of bits needed for representing one data point (e.g. if value of data points are of type double, f is usually $8 \times 8 = 64$).

Considering that b is always less than 10, and τ is about percent, and f is usually larger than 4 bytes, the index file is reduced to smaller than 1/10 of the original data file.

Input: A list of series data $\mathcal{O}$ (where each element v of $\mathcal{O}$ is a series of values $v_1, v_2, \ldots, v_n$, and v_i is a real number), the number of bits assigned for approximating each value b, and tolerance ϵ.
Output: the DDR of $\mathcal{O}$

let $\beta = (\beta_1\beta_2 \ldots \beta_n)_2$ be a n-bit pattern, α_0 be a b bit integer.
begin
  foreach $v = v_1, v_2, \ldots, v_n$ of $\mathcal{O}$ **do**
    $\beta \leftarrow 0,\ \alpha'_0 \leftarrow 0;$
    for $i = 1, 2, \ldots n$ **do**
      if $v_i - \alpha'_{i-1} \times h > h + \epsilon$ **or** $\alpha_{i-1} \times h - v_i > \epsilon$ **then**
      {exactly the condition shown in Equation 4.2}
        $\beta_i \leftarrow 1, \alpha_i \leftarrow \lfloor v_i \times 2^b \rfloor, \alpha'_i \leftarrow \alpha_i$
      else
        {α_i is omitted}
        $\alpha'_i \leftarrow \alpha'_{i-1}$
      endif
    endfor
    Append β and α to DDR
  endforeach
end

Algorithm 3: Constructing DDR index file

Using this algorithm, we know that each entry of the DDR index can be found easily from the beginning. In particular, the first entry has an n-bit header β, followed by the same number of quantized values as the number of none zero bits in β. Each quantized value α_i has length of b bits. The second and the following entries are constructed one by one in the same way. Although DDR is not suitable for random access that is necessary in the refinement phase (Algorithm 5), it does not hurt overall performance of our algorithms, since it is used only for scanning to generate candidates (Algorithm 4).

Input: A series data q as a query, $q = (q_1, q_2, \ldots, q_n)$.
Output: The list s of candidate identifiers of the k-NN to q.

let $UpBound[i]$'s be upper bound of candidates;
begin
    id $\leftarrow$ 1;
    $UpBound[1], UpBound[2], \ldots, UpBound[k] \leftarrow \infty$
    foreach entry α in DDR **do**
        Compute the upper bound D_{UB} and lower bound D_{LB}
            between α and q by Equation 4.3 and 4.4;
        if $|s| < k$ **or** $D_{LB}(q, v) \leq UpBound[k]$ **then**
            Insert the pair of id and D_{LB} to s,
                keeping s in the asscending order of D_{LB};
            Insert D_{UB} into *UpBound*,
                keeping it in assending order;
        endif
        id++;
    endforeach
end

Algorithm 4: Filtering phase using DDR

In the refinement phase (Algorithm 5), the candidates in the list s output by Algorithm 4 are checked. By id, the original series data is identified and its distance to the query q is computed. Each comparison necessitates a random access. There is no need to access objects corresponding to all elements in s. Note that s is sorted in the ascending order of the lower bound of its elements. When the remaining elements have greater bound than the real distance of k-th answer found so far, the search terminates, because we guarantee that no possible nearer objects will be found.

Input: s that is the output of Algorithm 4.
Output: The k-NN subset of q from $\mathcal{O}$.
ans=($\langle id[1], dst[1]\rangle, \langle id[2], dst[2]\rangle, \ldots, \langle id[n], dst[n]\rangle$)

begin
  $dst[1], dst[2], \ldots, dst[n] \leftarrow \infty$
  foreach time series data v in s **do**
    if $D_{LB}(q, v) > dst[k]$ **then**
      return *ans*
    endif
    if $D(q, v) < dst[k]$ **then**
      Insert v into *ans*,
        keeping it in assending order of *dst*;
    endif
  endforeach
end

Algorithm 5: Refinement phase

4.5 Experimental Evaluation

In this section we will demonstrate the superiority of DDR. The response time is counted by the number of page accesses.

We performed all tests over a range of query lengths. In our experiments, 10-NN queries are used for queries of length 256, 512, 1024, respectively. The query sequences are extracted from the datasets randomly. L_2 is used as distance function. We use a page size of 8KB in all our experiments.

For testing whole matching, we converted subsequences by sliding a "window" of query length n along v. Though this causes storage redundancy, it simplifies the notation and algorithms. The dataset is a *"relatively clean and uncomplicated"* electrocardiogram taken from the MIT Research Resource for Complex Physiologic Signals [30]. Totally, there are $100,000$ objects in the dataset. As mentioned earlier, data objects are normalized. That is,

any data point has a real value between 0 and 1.

In the figures displaying experiments, we used *reduced dimensions* and *index dimensions* to stand for the dimensions omitted and stored, respectively. This indicates that the sum of the two numbers is the dimensionality of the original data. Fixing the original dimensionality, smaller number of index dimensions means higher possibility of reduction.

4.5.1 Experimental Result: Reduction of Dimensionality

Recall that the number of bits b assigned for approximation and the efficiency of the index file is a trade-off. Using more bits enhances the accuracy of the index file (less false drops in the filtering phase) but makes the index file grow rapidly. The increase comes not only from the storage of more bits but also from the decreased possibility of omitting value. More bits lead to more divisions of grids, but needless to say, larger index file increases the cost of scanning. We investigated the relationship between reducing ratio of dimensionality and b in Figure 4.10. We found that the effeciency becomes maximum(efficiency) when b is assigned 4.

The ratio of dimensions being reduced determines the efficiency of DDR . A larger error tolerance ϵ leads to more reduction of dimensions. The demerit is that the bounds become looser, which results in less filtering efficiency. According to our experiments, the best efficiency appears when ϵ is assigned to $h/2$. Figure 4.11 illustrates the reduction effect with respect to dimensionality under the parameter value $\epsilon = h/2$. In this figure, it can also be seen that the reduction effect appears more significant for higher dimensional data.

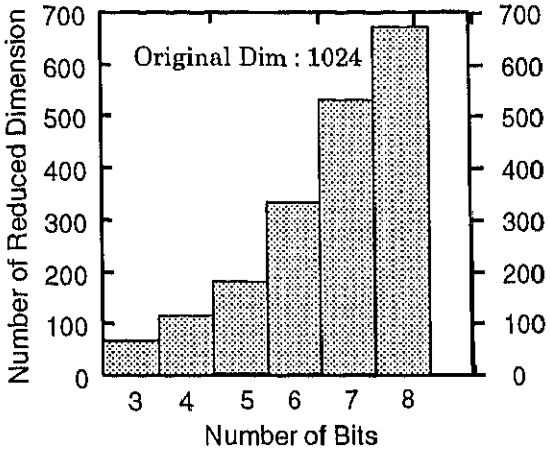

Figure 4.10: Average Dimensionality Reduced. The original dimensionality is 1024, and the error tolerance is half of grid's height. The number of dimensions is almost proportional to b.

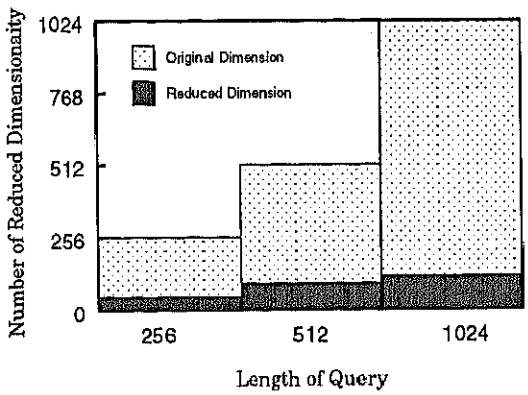

Figure 4.11: Average dimensionality reduced corresponding to the case of $\epsilon = h/2$ and $b = 4$. Reduction is more effective for higher dimensionality.

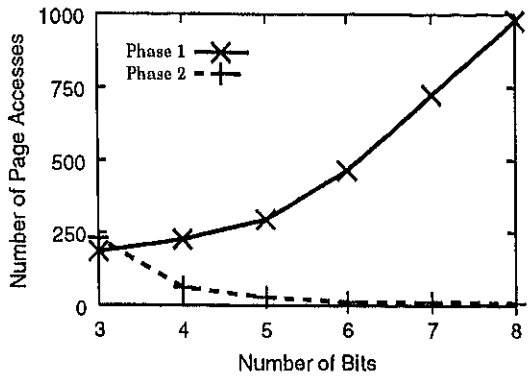

Figure 4.12: Total number of I/O page accesses. The query length is 1024, and the number of page accesses of phase 1 is scaled to 1/10 of the original values (weight factor 10 has been taken in tocount for comparing with that of phase 2).

In the filtering phase, scanning on a smaller index file leads to decreasing in I/O. On the other hand, looser bound brings more false drops, which leads to more random accesses in the refinement phase and increases the total response time. Figure 4.12 indicates the numbers of page accesses in filtering phase and refinement phase respectively. Considering the argument that random accesses are much more expensive than sequential accessed, we adjust the cost by introducing a weight of 10. That is, one random page I/O in the refinement phase is counted as equal to ten pages I/O in the filtering phase. Generally, it can be seen that when b is 4, the total number of page accesses of the two phases is the lowest.

4.5.2 Experimental Result: Comparison on number of page accesses

The comparisons with competing methods are illustrated here. Experiments are on a dataset of electrocardiogram. As the targets to compare, we implemented three indexing approaches other than our DDR-index. They are DFT-index, PAA-index, and linear scan.

In DFT-index and PAA-index technique, we reduce dimensionality of the time series data using DFT and PAA as described in Section 4.1.1. An index structure is built on the index dimensions for each technique. We adopt the SR-tree as the index structure. The k-NN search algorithm suggested by [34] is used. Two kinds of page accesses are considered: the one in accessing the index structure for getting the candidates of answer, and the one in accessing the full time series for refining the candidates.

Unlike other methods, our approach needs the refinement phase which makes random page accesses. The number of in this phase is shown in Table 4.1. To make the comparison fair, the total number of our page accesses is not simply the sum of the numbers in the two phases. Instead, we times the number of page accesses in the refinement phase by 10 before adding the number of filtering phase. Figure 4.13 illustrates the comparison for three kinds of queries lengths 1024,512 and 256. As can be seen from this figure, DDR retrieves much less pages than other methods. The difference is too wide so we have to plot in logarithm scale for comparing them in the same figure. Even taking the weight factor 10 into consideration, the small numbers ensure us that effects of the first phase are much more significant. The length b's of bit patterns quantizing coordinates are assighned to 6, 5 and 4 for original dimensionality 256, 512 and 1024, respectively.

Table 4.1: Number of Page Accesses in Phase 2

Dimension	256	512	1024
Number of page	49	39	16

Though DDR approach costs more CPU computation time than the other 3 approaches

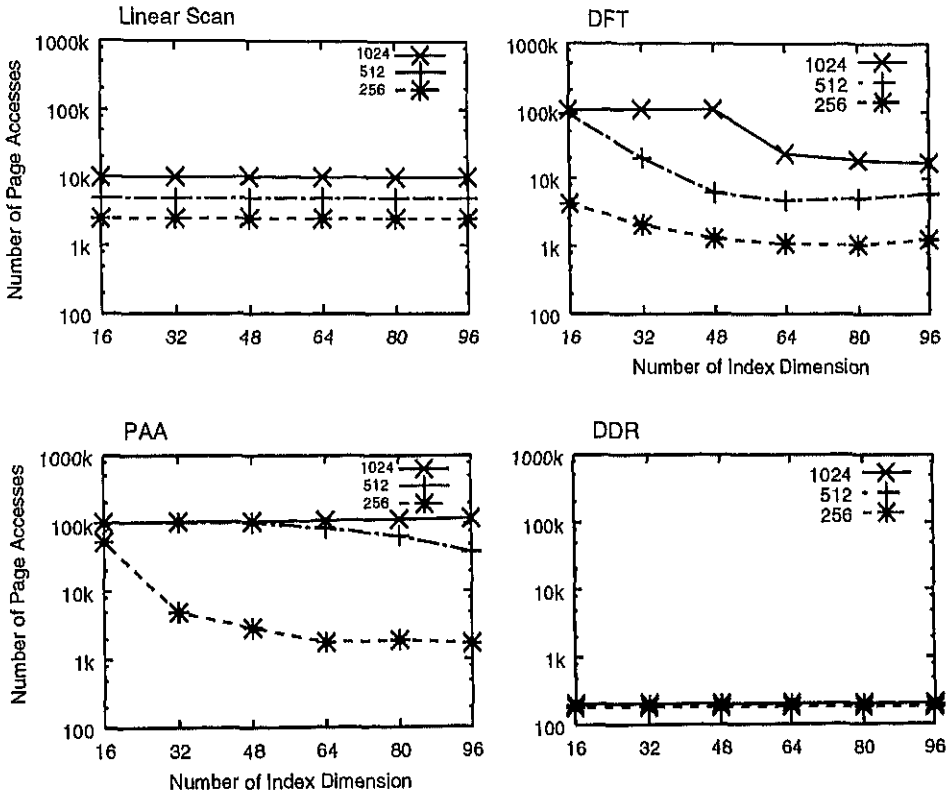

Figure 4.13: Comparison on number of page accesses. Linear Scan and DDR are plotted in the same form as DFT and PAA for comparison. In fact, no dimensionality reduction happens in linear Scan, while the reduction in DDR is not fixed but depends on data.

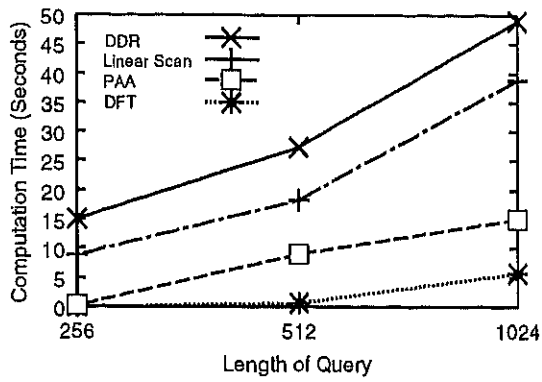

Figure 4.14: Comparison on number of CPU cost. As a sample of DFT and PAA, their index dimensions are 64.

as shown in Figure 4.14, DDR preserves the superiority as shown by the fact that the page accesses in DDR is $1/5 \sim 1/70$ of the other approaches.

4.6 Conclusions

we proposed an approach of dimensionality reduction for time series data. Combining with the technique of quantization, a grid-based indexing is designed and implemented. Experimental results are compared with some related works and confirm the efficiency of our method. As the future work, though the parameter b and tolerance ϵ are tuned to perform most effectively in the experiments, systematical determination is desirable.

So far in this thesis, we focussed on developing index structures and dimensionality reduction technique to handle high dimensional data. In the next chapter, we address the challenge of visualization of high dimensional datasets.