Subseries Join and Compression of Time Series Data Based on Non-uniform Segmentation

by

Yi Lin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A time series is composed of a sequence of data items that are measured at uniform intervals. Many application areas generate or manipulate time series, including finance, medicine, digital audio, and motion capture. Efficiently searching a large time series database is still a challenging problem, especially when partial or subseries matches are needed.

This thesis proposes a new definition of subseries join, a symmetric generalization of subseries matching, which finds similar subseries in two or more time series datasets. A solution is proposed to compute the subseries join based on a hierarchical feature representation. This hierarchical feature representation is generated by an anisotropic diffusion scale-space analysis and a non-uniform segmentation method. Each segment is represented by a minimal polynomial envelope in a reduced-dimensionality space. Based on the hierarchical feature representation, all features in a dataset are indexed in an R-tree, and candidate matching features of two datasets are found by an R-tree join operation. Given candidate matching features, a dynamic programming algorithm is developed to compute the final subseries join. To improve storage efficiency, a hierarchical compression scheme is proposed to compress features. The minimal polynomial envelope representation is transformed to a Bézier spline envelope representation. The control points of each Bézier spline are then hierarchically differenced and an arithmetic coding is used to compress these differences.

To empirically evaluate their effectiveness, the proposed subseries join and compression techniques are tested on various publicly available datasets. A large motion capture database is also used to verify the techniques in a real-world application. The experiments show that the proposed subseries join technique can better tolerate noise and local scaling than previous work, and the proposed compression technique can also achieve about 85% higher compression rates than previous work with the same distortion error.

Acknowledgments

First of all, I wish to express my deep gratitude to my supervisor, Dr. Michael McCool for his valuable advice, brilliant inspiration, and persistent support. I would like thank my Ph.D. committee, Dr. Aijun An, Dr. Liang-Liang Xie, Dr. Ming Li, and Dr. Stephen Mann, for their efforts to review my thesis, and for their valuable suggestions and great inspiration. I also would like to thank Dr. Bill Cowan, for his guidance and help at the early stage of my study at the University of Waterloo.

I am grateful to Dr. Eamonn Keogh from the University of California, Riverside, for providing me the UCR Time Series Achieves as the testing data for this thesis. I would like to thank the Graphics Lab, Carnegie Mellon University, for offering me a large motion capture database used in this thesis. I want to thank the RapidMind Inc. for providing me a state-of-the-art parallel software development environment, RapidMind Multi-Core Development Platform, which was used in my prototype system implementation. Also thanks belong to Philippe Beaudoin and Pierre Poulin from the University of Montréal, and Michiel van de Panne from the University of British Columbia, for providing comparison data for my motion compression work.

I owe many thanks to my school fellows who have provided many interesting discussions and insights on my work, especially, Yu Chen, Gabriel Esteves, Elodie Fourquet, Jiwen Huo, Ghulam Lashari, Andrew Lauritzen, Yingbin Liu, Stuart Polloc, Zheng Qin, Jie Xu, and Daming Yao.

This thesis is dedicated to my parents, who have always been there for me and encouraged me to pursue my dreams.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Time series are ubiquitous. Many applications generate time series data, such as finance, medicine, music, and motion capture. Research into efficient methods for time series retrieval and compression is becoming increasingly important as the volume of data increases. The main topics in this thesis are time series retrieval, especially the problem of subseries join, and time series compression. In this chapter, background materials are first reviewed on time series retrieval and compression. Then an overview of the techniques proposed in this thesis is presented and the specific contributions of this thesis are listed. An outline of the thesis is given at the end of this chapter.

1.1 Subseries Join and Compression

Time series retrieval belongs to the area of information retrieval. The purpose of information retrieval is to identify and to extract information that satisfies a user's needs from a large collection of data. A retrieval algorithm finds matching information in a collection given a description of a need, and extracts the pieces of information that are relevant to that need. Generally, retrieval methods can be classified into two categories [SBNW96], pattern-based retrieval and content-based retrieval.

The first category is *pattern-based retrieval*, which uses some semantic analysis or

knowledge, such as annotations or labels, to state the requirements of the needed information. The word "pattern" means a set of constraints that must be satisfied when searching a dataset. The pattern can be in the form of a high-level language description, such as "find all the running motions in a motion capture database", or can contain more detailed information, such as "find all running motions in which the left foot is lifted 10cm higher than the right foot in two successive cycles". No matter what the pattern is, it has to be converted to a mathematical description that is recognizable to the computer. To answer pattern match queries, as long as the data contain the specified pattern, they will be retrieved, no matter where and how the pattern appears. One example of pattern query retrieval is the use of *Structured Query Language (SQL)*. Pattern-based retrieval requires complete and accurate domain knowledge for pattern extraction. This requires either intensive human-assisted annotations or extremely (perhaps impractically so) sophisticated automatic content analysis. Sometimes, an approximation can be used in constructing a useful index. Some examples are automatic speech recognition and audio retrieval systems that recognize the words uttered in an audio stream.

The second category is *content-based retrieval* (or *example-based retrieval*), which uses some physical description, or comparison with examples from a similar medium, in order to describe the needed information. For time series, content-based retrieval finds pieces of time series data that have similar shapes to a given example time series. For time series data, patterns are more difficult to extract or to describe than symbolic (or text) data, because time series are samples from continuous signals. In contrast, examples are plentiful and patterns can be built up by considering similarity to such examples. Therefore, content-based retrieval is useful for many real-world applications. For instance, the user may want to find companies whose stock prices move similarly, or to find cases in the past that resemble the sale patterns in this month's data.

To evaluate a retrieval algorithm, there are two criteria: *accuracy* and *efficiency*. Accuracy specifies that the retrieval algorithm returns no false dismissals, relative to

some "gold standard". *False dismissals* (also called *false negatives*) occur when data that should be retrieved are not retrieved. Correspondingly, *false alarms* (also called *false positives*) occur when data that should not be retrieved are retrieved. Accuracy is based on the definition of similarity measure and the gold standard is usually an exhaustive search (which is, of course, not practical) using that similarity measure. Unfortunately, an all-purpose similarity measure for different data, domains, and tasks can be difficult to establish.

Efficiency requires the retrieval algorithm to find the correct answer as rapidly as possible. To search a large database, an index usually needs to be constructed, because sequentially scanning the database using exhaustive search is too slow to be practical for most applications. An index can narrow the search to a small set of candidates of the database. A search mechanism can be exact (allowing no false dismissals or false positives) or approximate (allowing some number of either). An approximate retrieval algorithm should measure the level of user satisfaction and permit a balanced tradeoff between accuracy and efficiency.

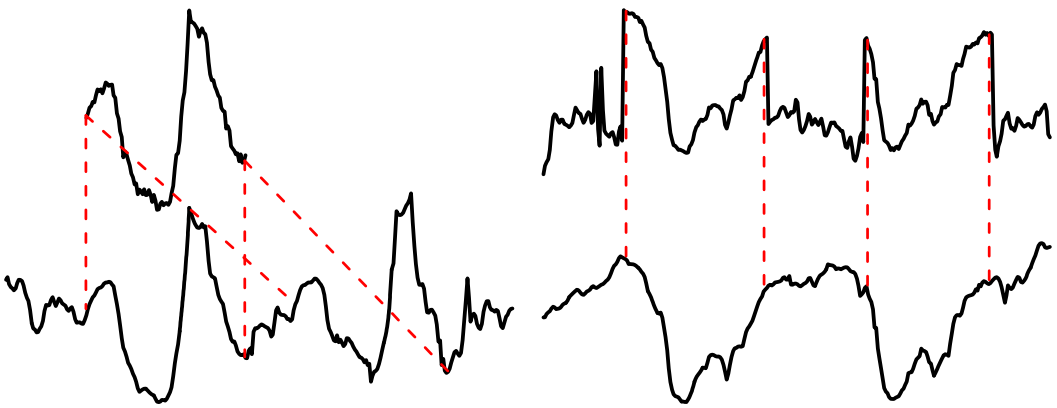
The problem of *whole matching* for time series has been studied for many years. More recent work also studies the problem of *subseries matching*. Whole matching finds time series in a dataset that are similar to a given query time series (see Definition 2.8). Subseries matching finds similar subseries of a time series in a dataset to a given query time series (see Definition 2.9).

Recently, the problem of *subseries join* has been implicitly identified [Keo06b]. The goal of subseries join is similar to that of classic relational database join, which combines all the elements of two data sources that satisfy a similarity criterion [YKM+08]. Subseries join finds similar subseries in one time series dataset to subseries of another time series dataset (see Definition 2.10). Compared with subseries matching, which is a one-to-many, asymmetrical operation, subseries join is a many-to-many, symmetrical operation. This is illustrated in Figure 1.1. Subseries join is potentially useful for many data mining applications, including clustering [Elk03], classification [XKS+06],

anomaly detection [WKX06], rule discovery [MRK+07], and motif discovery [YKM+07]. Figure 1.2 shows the conceptual definition of subseries join.

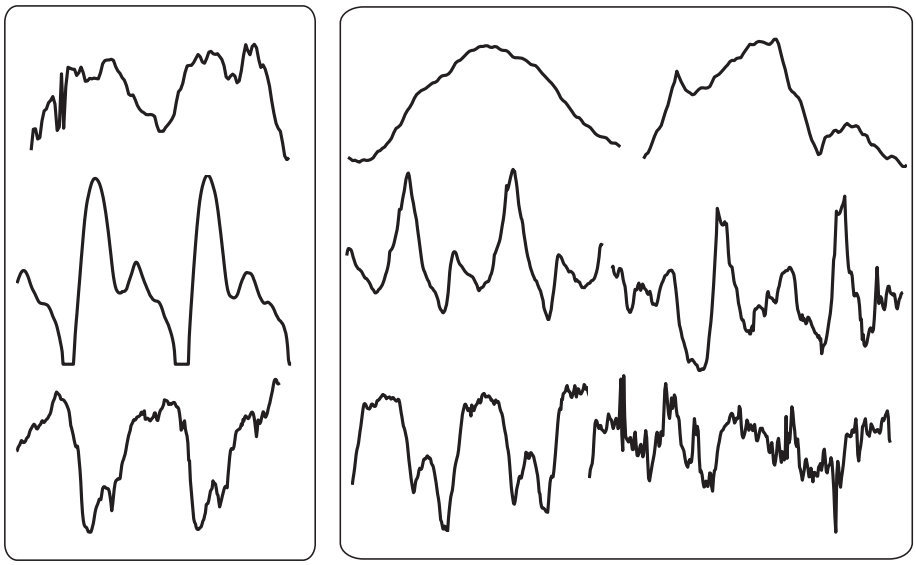
Previous methods based on similarity measures and indexing for whole matching and subseries matching do not work for subseries join. This thesis proposes a solution to subseries join based on the formalization given in Definition 2.10. Subseries join is a generalization of subseries matching, just as the proposed definition of subseries matching is a generalization of whole matching. Therefore, the proposed solution also works for whole matching and subseries matching.

A representation is used for indexing that also leads to compressed storage, so the proposed approach also allows searching compressed time series datasets. The purpose of data compression is to convert an input representation to an output representation that has a smaller size. A compression method encodes the same information more compactly by removing redundancy. Basically, compression methods can be classified into two categories: *lossless* and *lossy*. Lossless compression methods have zero error but cannot achieve high compression rates, especially in the presence of noise. Lossy compression methods can achieve higher compression rates but always involve a trade-

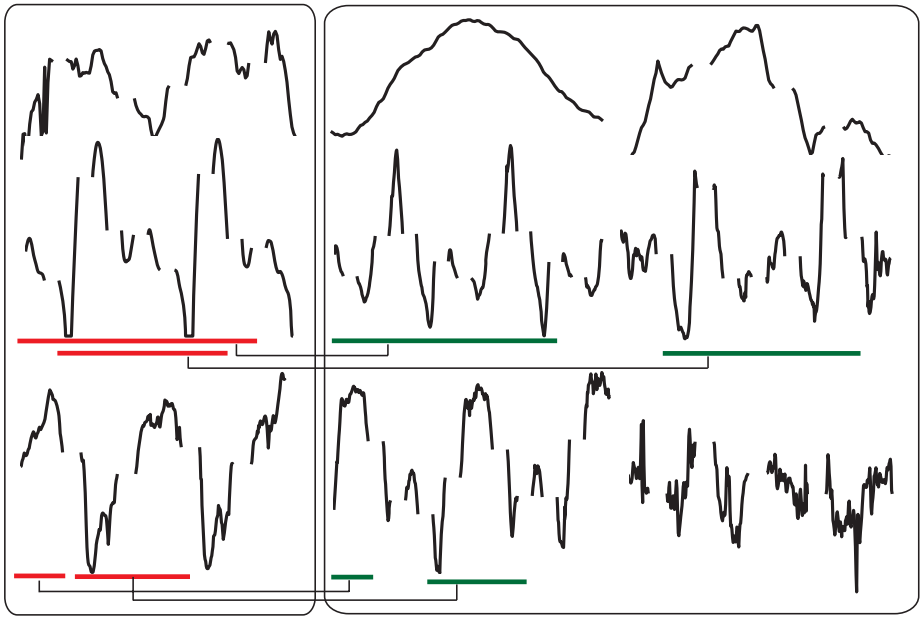


(a) Subseries matching (b) Subseries join

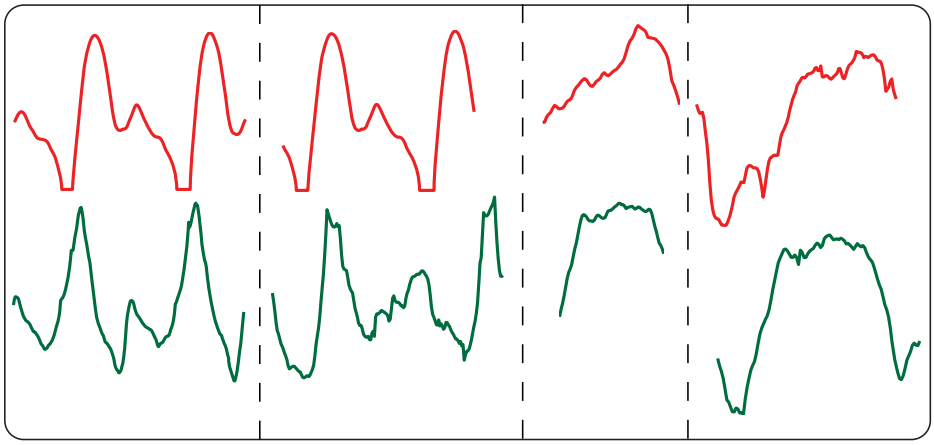
Figure 1.1: Subseries matching *vs.* subseries join.



(a) Two time series datasets.



(b) Time series are non-uniformly segmented and candidates are generated by matching sequences of segments using dynamic programming.



(c) The result is a set of pairs of matching subseries that satisfy a similarity threshold and are maximal-length.

Figure 1.2: Conceptual definition of subseries join.

off between compression rate and error. Lossy compression methods should remove redundancy as much as possible without discarding important information. They cannot guarantee that the decompressed data will be identical to the original data but attempt to avoid important differences. However, what information is "important" can be content-dependent and domain-dependent. In some domains, such as music or motion capture, an aspect of the data can be considered "unimportant" if its removal is perceptually undetectable.

1.2 Research Overview

This thesis focuses on investigating content-based retrieval and lossy compression techniques for time series. For time series retrieval, a novel definition is proposed for the problem of subseries join whose definition will be presented in Section 2.1.1. Figure 1.3 gives a graphical overview of the proposed solution to solving the problem of subseries join. The proposed solution consists of a number of stages. Specifically, the original

time series data are first transformed to a hierarchical feature sequence. To obtain this hierarchy, the time series are smoothed using an anisotropic diffusion process. The smoothed data is then segmented at discontinuities that are identified by the zero-crossings of the second derivatives after anisotropic diffusion. One of the properties of anisotropic diffusion is that it pins the locations of the zero-crossings of the second derivatives across scales, resulting in a stable hierarchy of segments. A segment is then approximated and represented by a minimal polynomial envelope (and possibly other additional derived parameters that reflect the properties of the segment), which is referred to as a *feature* in this thesis. In this way, each segment is mapped to a lower-dimensional feature point. All the features are indexed in an R-tree in such a way that distances between segments are equivalent to distances between feature points.

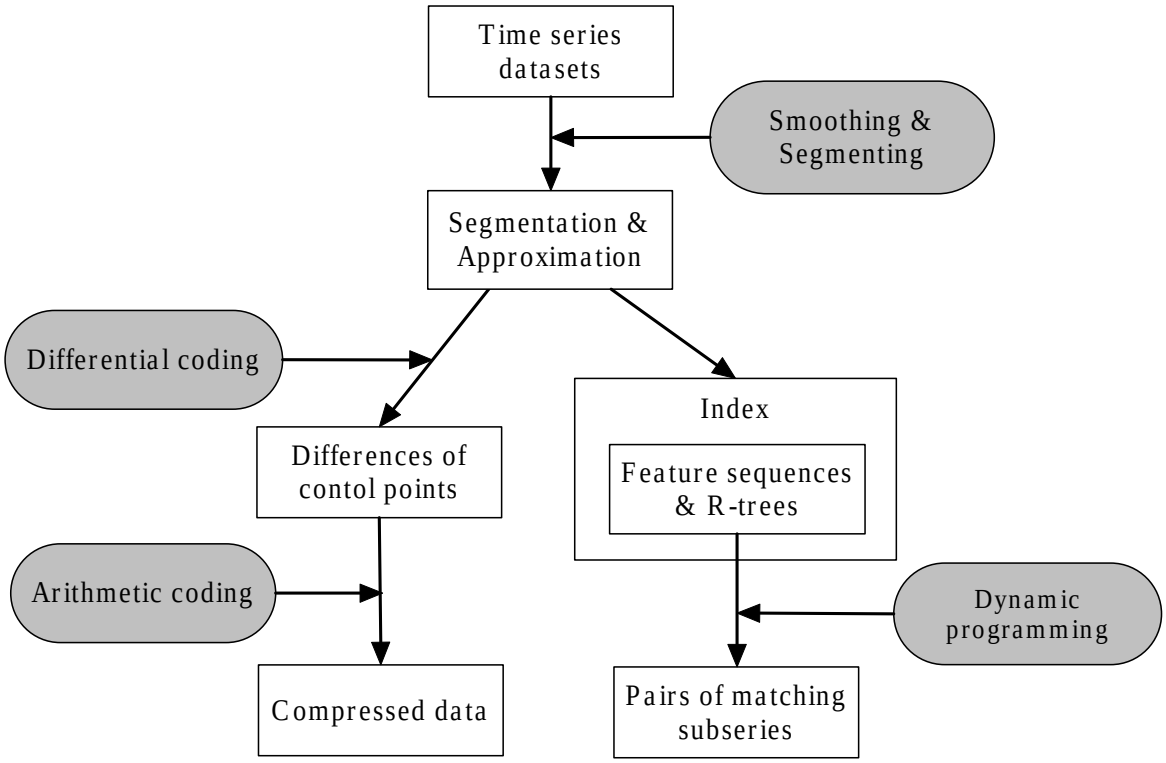


Figure 1.3: Overview of the proposed techniques.

An R-tree join operation is then performed on the R-tree indexes of two datasets. The result of this R-tree join is the set of all pairs of features from the two datasets that are closer to each other than a predefined distance threshold relative to the proposed distance metric. Once a set of such candidate pairs is found, they are joined into maximal-length subseries using dynamic programming.

A compression scheme is developed based on the same segmentation and representation. Each segment is approximated using a Bézier spline. The spline control points generated from the Bézier spline approximation are differentially and hierarchically encoded. This compression scheme is based on the fact that the spline control points in one scale can be approximated by a subdivision of the spline in a coarser scale. The differences are then encoded using arithmetic coding.

The proposed approach requires minimum domain-related knowledge, so that few changes are needed to apply the approach to different types of time series. The effectiveness of the proposed techniques are empirically confirmed using various publicly available datasets. In particular, to test the effectiveness of the proposed approach in a specific application, the approach is applied to a large motion capture database.

1.3 Contributions

The major contributions of this thesis are as follows:

1. A new definition of subseries join is proposed and a novel solution is developed to computing it. Recently, the problem of subseries join has been implicitly identified. We give a more formal definition, so that subseries join is a generalization of subseries matching and whole matching. Solutions to subseries join can also solve the problems of subseries matching and whole matching.

2. A new hierarchical representation is generated using an anisotropic diffusion scale-space analysis and a non-uniform segmentation method. This non-uniform

segmentation method divides time series into segments based on its intrinsic structure, which avoids cutting features as uniform segmentation methods do. A minimal polynomial envelope and other additional parameters are used to represent each segment in a reduced-dimensionality space, suitable for indexing and compression. The minimal polynomial envelope can achieve tighter lower bounds than previous lower bounding methods. The hierarchical representation and index can automatically be adapted to fit into any memory system.

3. A dynamic programming algorithm is proposed to compare and align feature sequences over a scale space. Novel distance functions and gap penalty functions are also defined to adapt to time series. This algorithm can tolerate both impulsive noise and additive noise, which are not tolerated well by previous distance functions.

4. A compression scheme is proposed to compress time series data. This algorithm can achieve higher compression rates than previous work. By using anisotropic diffusion, data are smoothed so that the influence of noise is avoided on the compression rate. Noise can adversely and often severely affect the compression rate of competing schemes. At the same time, important discontinuities are retained. A differential coding method is presented to do further compression based on the hierarchical structure. Updating the database will not influence the compression of unchanged data. It is convenient to evaluate the data at any point in time directly from the compressed representation, and compressed databases are still searchable. Finally, the decompression is fast enough for real-time applications.

1.4 Outline

Background knowledge, previous work, and underlying technology are reviewed in Chapter 2. Related definitions are also given in this chapter. In Chapter 3, a new representation, based on a non-uniform segmentation that maps a time series into a non-uniform feature sequence, is described. An indexing and subseries join method based on the hierarchical feature sequences used in this representation is presented in Chapter 4. The evaluation of the proposed approach and experimental results are shown in Chapter 5. Experiments for subseries join and compression of a large motion capture database are presented in Chapter 6. At the end of the thesis in Chapter 7, main contributions and results of this thesis are summarized and possible directions for future work are identified and discussed.

Chapter 2

Background and Related Work

In this chapter, background information on time series, time series retrieval, and time series compression is introduced. Specifically, for time series retrieval, this chapter discusses on the problem of subseries join based on a new definition. Previous work on topics related to time series retrieval is also reviewed including similarity measures, dimensionality reduction strategies, and indexing methods.

2.1 Background

A time series is composed of a sequence of values measured from a continuous signal. The term "time series" is often used to refer to any such sampled data set with one independent variable, whether or not that independent variable is time. A time series is a sequence of samples representing value(s) at specific points in time. If the sample is a single value (a number), then the time series is called a *single-channel time series*. If the sample is a vector of values (a vector of numbers), then the time series is called a *multi-channel time series*. Examples of time series data include financial data, scientific measurements, weather data, music data, and motion capture data. A multi-channel time series has multiple data streams usually sampled at the same rate. For example, a motion capture data records the angular values of each joint in a skeleton, and records

each angle at moments that are with synchronized the other joints. Figure 2.1(a) and Figure 2.1(b) show examples of a single-channel time series of stock data[1] and a multi-channel time series of motion capture data.

Note that our interpretation of time series includes the assumption that they are sampled from a continuous signal. We assume, in particular, that these samples can be interpolated and filtered as in other areas of signal processing, and that the samples are an imperfect digital representation of some underlying continuous-time signal. The formal definitions of time series and uniform time series are given in Definition 2.1 and 2.2. This thesis only considers uniform time series. The theoretical parts of the thesis focuses on single-channel time series first. Chapter 6 will show how the proposed techniques can be extended to multi-channel time series.

Definition 2.1 *A time series X is a sequence of possibly vector-valued data that are sampled at successive points in time, denoted by*

$$X = ((x_1, t_1), \ldots, (x_n, t_n)), \tag{2.1}$$

where n is the number of sample timestamps and $t_{i+1} > t_i$ for all i. The element x_i is a vector of size k sampled at timestamp t_i, i.e.,

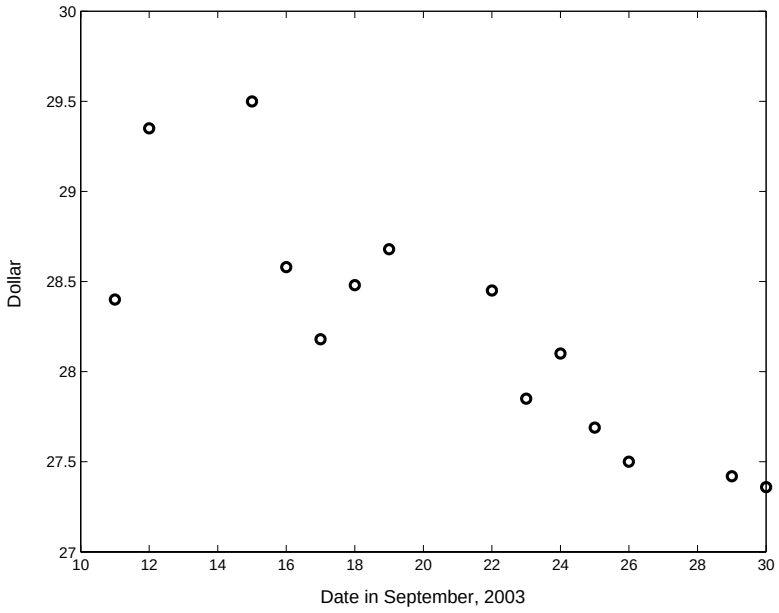
$$x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k}). \tag{2.2}$$

If $k = 1$, then X is a single-channel time series. If $k > 1$, then X is a multi-channel time series.

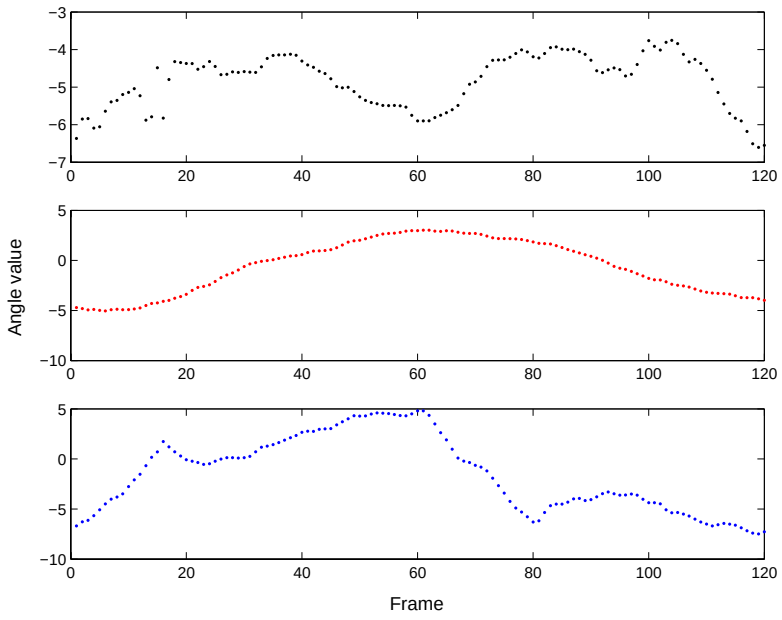
Definition 2.2 *A uniform time series X is a sequence of possibly vector-valued data that are sampled at uniform intervals. That is, Definition 2.1 with the additional constraint that*

$$t_i = iT + T_0, \tag{2.3}$$

[1]Stock prices are recorded at daily intervals excluding weekends and holidays.



(a) Single-channel time series: the high prices of a stock on September 11 $\sim$ 30, 2003, excluding three weekends.



(b) Multiple-channel time series: joint angles of a human motion at three degrees of freedom.

Figure 2.1: Examples of real-world time series.

where T_0 is the start time and T is the sampling period.

The basic notations used in this thesis are summarized in Table 2.1.

Table 2.1: The basic notations used in this thesis.

Symbol	**Description**
X	The time series, $X = (x_1, x_2, \ldots, x_i, \ldots, x_n)$
$\#X$	The number of elements in X
$X[i:j]$	The subsection of X from the i^{th} element to the j^{th} element
$\mathbb{D}$	The dataset of time series
$\#\mathbb{D}$	The number of time series in $\mathbb{D}$
W	The vector of weights for different channels, $W = (w_1, w_2, \ldots, w_c)$

2.1.1 Time Series Retrieval

Given a query, a retrieval method finds data sequences that satisfy the requirements of the query. Content-based retrieval is based on a "similarity" between a query and the data. The similarity is usually defined as a distance function that reflects how far the data is from the query.

For many data mining applications, finding similar subsections of time series data to a query sequence is of prime importance. Note, however, that in previous work the word "subsequence" has been used to describe any sub-part of a time series. Unfortunately, the term subsequence is already in wide use to describe *discrete* sequences of data such as strings and DNA sequences. In contrast, time series are often interpreted as samples from continuous signals. Therefore, to avoid confusion in this thesis the word "subseries" will be used to describe sub-parts of time series.

Definition 2.3 ***Subseries****: Given a time series*

$$X = (x_1, x_2, \ldots, x_n),$$

a subseries X' of X is

$$X' = (x_i, x_{i+1}, \ldots, x_{i+n'-1}),$$

where $1 \leq i \leq n$ and $1 \leq n' < n - i$.

Note that under the definition a subseries in Definition 2.3 is a continuous segment of a time series and does not include gaps. However, for impulsive noise immunity later on a definition of discrete subsequences that *do* allow for gaps is also used.

Definition 2.4 ***Subsequence****: Given a sequence*

$$A = (a_1, a_2, \ldots, a_n),$$

a subsequence A' of A is

$$A' = (a_{q(1)}, a_{q(2)}, \ldots, a_{q(n')}),$$

where $1 \leq q(i) \leq n$, $1 \leq q(n') \leq n$, and $q : \{1, \ldots, n'\} \mapsto \{1, \ldots, n\}$ is strictly monotonic and injective.

Similarity between time series is usually defined in terms of a *distance function* that reflects how far one time series (or subseries) is from another time series (or subseries):

Definition 2.5 ***Distance function****: A function d from pairs (X, Y) to the real numbers is a distance function if it satisfies the following conditions:*

Non-negativity: $d(X, Y) \geq 0$.

Identity: $d(X, Y) = 0$ *if and only if* $X = Y$.

Symmetry: $d(X, Y) = d(Y, X)$.

Minimizing a distance function results in maximal similarity. An ϵ-similarity is defined as a test relative to a threshold ϵ:

Definition 2.6 ***ϵ-similarity****: Given any two time series X and Y, we say X is ϵ-similar to Y if $d(X, Y) \leq \epsilon$, where ϵ is a predefined threshold value.*

We call X an ϵ-match to Y. Distance functions are also sometimes called *similarity measures*.

A distance function should only be called a *distance metric* if it satisfies the triangle inequality as well:

Definition 2.7 ***Distance metric****: A distance function d is a distance metric if it also satisfies the triangle inequality for all X, Y, and Z:*

Triangle Inequality:

$$d(X, Z) \leq d(X, Y) + d(Y, Z). \tag{2.4}$$

We can rewrite Equation 2.4 as $d(X, Y) \geq d(X, Z) - d(Y, Z)$. The value $d(X, Z) - d(Y, Z)$ can be treated as a lower bound on $d(X, Y)$, if $d(X, Z)$ and $d(Y, Z)$ are known. This is the basic principle that most distance-based indexing structures follow, such as VP-trees [Uhl91], M-trees [CPZ97], SA-trees [Nav02], and OMNI-family access methods [FTJF01].

Sometimes, instead of a distance function, a *scoring function* is used as a similarity measure. With a scoring function, a higher value indicates greater similarity.

The definition of the distance function (or metric) depends on the user, the domain, and the task. There does not exist an all-purpose similarity measure. In other words, the definition of similarity is subjective. However, the *Euclidean distance* metric (root-mean-square) is often used since other metrics can often be implemented in terms

of it using a suitable transformation of the data. *Dynamic Time Warping (DTW)* is another widely used distance function that is useful when time series need to be aligned with each other. However, neither the Euclidean distance nor DTW can be used in subseries matching or subseries join. The Euclidean distance requires the time series to have the same lengths, while DTW must match every element in each time series (whole matching). For example, with both distance functions matching five heartbeats in one cardiogram to six heartbeats in another cardiogram is meaningless.

Two kinds of time series matches are now defined, whole and subseries, in terms of ϵ-similarity:

Definition 2.8 ***Whole series match****: Given a time series X and a time series Y, if $d(X, Y) \leq \epsilon$, then X is a whole match to Y.*

Subseries match is a generalization of whole match:

Definition 2.9 ***Subseries match****: Given a time series X and a set (database) of time series $\mathcal{Y}$, the subseries match $\mathcal{M}$ is the set of all subseries $Y_{i,k} \subseteq Y_k$ for all $Y_k \in \mathcal{Y}$ such that $d(X, Y_{i,k}) \leq \epsilon$ and for which there does not exist any $Y'_{i,k} \supset Y_{i,k}$, where $Y'_{i,k}$ is longer than $Y_{i,k}$ and contains $Y_{i,k}$ as a proper subset and for which $d(X, Y'_{i,k}) < \epsilon$.*

The notation $A \subseteq B$ means that A is a subseries of B. Note that multiple matches are possible and one query may match different subseries from the same time series. This definition also eliminates any redundant matches that are parts of longer matches.

Finally, the definition of subseries join returns all pairs of subseries drawn from two datasets that satisfy the similarity threshold and are also maximal-length:

Definition 2.10 ***Subseries join****: Given two sets of time series $\mathcal{X}$ and $\mathcal{Y}$, the subseries join is the set of all pairs $(X_{i,k}, Y_{j,\ell})$ of subseries $X_{i,k} \subseteq X_k$ for $X_k \in \mathcal{X}$ and $Y_{j,\ell} \subseteq Y_\ell$ for $Y_\ell \in \mathcal{Y}$ such that $d(X_{i,k}, Y_{j,\ell}) \leq \epsilon$, and for which there does not exist any $X'_{i,k} \supset X_{i,k}$ and $Y'_{j,\ell} \supset Y_{j,\ell}$ where $X'_{i,k}$ is longer than $X_{i,k}$ and contains $X_{i,k}$ as a proper subset and where $Y'_{j,\ell}$ is longer than $Y_{j,\ell}$ and contains $Y_{j,\ell}$ as a proper subset for which $d(X'_{i,k}, Y'_{j,\ell}) < \epsilon$ or for which $d(X_{i,k}, Y'_{j,\ell}) < \epsilon$ or for which $d(X'_{i,k}, Y_{j,\ell}) < \epsilon$.*

Note that the result of a subseries join contains all subseries matches if one of the input datasets $\mathcal{X}$ is defined as the singleton set $\{X\}$. However, the subseries join also includes other partial matches from the subseries of the dataset to subseries of X. Therefore, to obtain a subseries match result exactly as defined above from a join, these additional partial matches should be filtered out.

When searching in a large dataset, there may exist many similar time series to the query. Based on how the number of results is constrained, retrieval methods can be classified into *range retrieval* (see Definition 2.11) and *k-Nearest Neighbor (k-NN)* retrieval (see Definition 2.12). The range retrieval and the k-NN retrieval methods can be easily converted to each other by selecting multiple ranges or different values of k.

Definition 2.11 ***Range retrieval****: Given a query time series Q, a range retrieval finds all the time series X in a dataset $\mathbb{D}$ that satisfy $d(Q, X) \leq \epsilon$.*

Definition 2.12 ***k-Nearest Neighbor (k-NN) retrieval****: Given a query time series Q, a k-Nearest Neighbor (k-NN) retrieval finds a subset $\mathbb{D}' \subseteq \mathbb{D}$ such that $\{\#\mathbb{D}' = k | \forall X \in \mathbb{D}', Y \in \mathbb{D} - \mathbb{D}', d(Q, X) \leq d(Q, Y)\}$. When $k = 1$, k-NN retrieval returns the best match.*

2.1.2 Time Series Compression

Compression of time series data can exploit redundant information across three dimensions: the temporal dimension, the space/scale dimension, and the time series clip dimension. Most compression approaches use a *decorrelation* step followed by a *coding* step. Decorrelation removes redundancy between different data elements. Coding exploits the differences in probability between different data values to reduce the average data rate. Specifically, high-probability data values can be coded with shorter codes than low-probability data values. Coding algorithms are relatively standard; most compression schemes differ primarily in their approach to decorrelation, since different data types have different kinds of redundancy.

Compression techniques can be classified into two categories: lossless and lossy. Lossless compression has zero error but cannot achieve high compression rates. Lossy compression techniques can achieve higher compression rates but always involve a tradeoff between compression rate and error. A lossy compression may take advantage of the properties of data, in which deleting some information in the data would not be detected by human perception. Non-symbolic time series can often be simplified in many ways without a noticeable degradation of quality. The goal of a lossy compression method is to achieve the best ratio between compression rate and error. The *compression rate* (or *compression ratio*) is a performance measure for compression.

Definition 2.13 *Given the input time series X and the reconstructed time series Y,*

$$\textit{compression rate} = \text{size}(X)/\text{size}(Y), \tag{2.5}$$

where size *is the number of bytes of storage required to contain the data.*

Compression error metrics usually vary with different types of data. One basic error metric that applies to any kind of real-valued data is the Euclidean distance metric, which is also called the *root mean square error (RMSE)* metric.

Definition 2.14 *Given the input time series X and the reconstructed time series Y, the root mean square error (RMSE) between X and Y is*

$$RMSE(X,Y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{2.6}$$

The weighted root mean square error (WRMSE) is

$$WRMSE(X,Y) = \frac{1}{n}\sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}, \tag{2.7}$$

where w_i is the weight assigned to each element.

2.2 Similarity Measures

A similarity measure reflects how alike the query and the retrieved data are. The similarity of two time series can be measured by a distance function. In this section, the most common and useful distance functions for time series are reviewed, including the Euclidean distance metric, the Dynamic Time Warping distance measure, and the Longest Common Subsequence method. Sequence alignment algorithms that measure distances between discrete sequences are also introduced.

2.2.1 Euclidean Distance Metric

The *Euclidean distance* metric is a special form of the L_p-norm distance metric. The L_p-norm is the simplest distance metric.

Definition 2.15 *Given two time series X and Y of the same length n, the L_p-norm distance metric between X and Y is*

$$L_p(X,Y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}. \tag{2.8}$$

If X and Y are multi-channel time series with c channels, then

$$|x_i - y_i| = \sum_{j=1}^{c} w_j (x_{i,j} - y_{i,j}). \tag{2.9}$$

The value w_j is the weight value assigned to the j^{th} channel. When $p = 1$, the L_1-norm metric is called the *Manhattan distance* or the *city block distance*. When $p = 2$, the L_2-norm metric is the well-known *Euclidean distance*.

The Euclidean distance needs linear computational cost in terms of time series length. However, it requires the two time series to have exactly the same length and it cannot handle local time shifting. *Local time shifting* occurs when one element in

one time series must be shifted along the time axis to match an element in the other time series, i.e., the two matched elements appear in different positions in the time axis. The word "local" means that not all the elements of one time series need to be shifted and all the shifted elements do not need to have the same shifting factor. By contrast, in *global time shifting*, all the elements are shifted along the time axis by a fixed shifting factor. The *shift-and-compare* method shifts the shorter time series to every possible offset of the longer time series and compares the two time series using an L_p-norm metric. Time series that have similar shapes often require local time shifting when they are matched. The L_p-norm cannot handle local time shifting, because it requires the exact position-by-position correspondence of elements in two time series, i.e., the i^{th} element of one series must be aligned with the i^{th} element of the other series. This problem can be solved by Dynamic Time Warping.

2.2.2 Dynamic Time Warping

Due to misalignment, two time series may look similar but may not be considered close in the Euclidean distance. As a solution to this, Myers and Rabiner [MR81] introduced DTW, which has been widely used in data mining [YJF98, KP00], gesture recognition [GD95], robotics [SOC99], speech processing [RJ93], manufacturing [GP95], and medicine [CPB+98]. *Dynamic Time Warping (DTW)* is a distance measure based on first computing an optimal alignment. It is more robust than the Euclidean distance when sequences may have different lengths or patterns that are out of phase in the time axis. By computing an alignment first, DTW handles these cases.

Given two time series $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, DTW aligns each element x_i of X $(1 \leq i \leq m)$ to one or more elements y_j of Y $(1 \leq j \leq n)$; also, every element of Y may be aligned to one or more elements of X. DTW performs dynamic programming on an $m \times n$ distance matrix, in which each matrix element (i, j) contains the squared Euclidean distance $d_E(x_i, y_j) = \sqrt{(x_i - y_i)^2}$. From the distance matrix, a

warping matrix WM is constructed by the recurrence

$$WM(i,j) = d_E(x_i, y_j) + \min \begin{cases} WM(i-1, j-1) \\ WM(i, j-1) \\ WM(i-1, j). \end{cases} \tag{2.10}$$

The initial conditions of this recurrence are $WM(0,0) = 0$ and $WM(i,0) = WM(0,j) = \infty$. The square root of $WM(m,n)$ defines the DTW distance:

$$DTW(X,Y) = \sqrt{WM(m,n)}. \tag{2.11}$$

The global optimal alignment is $U = (u_1, \ldots, u_i, \ldots, u_k)$, where $\max\{m,n\} \leq k \leq m+n-1$, gives a contiguous set of matrix elements that defines a mapping between X and Y. The warping path is typically subject to the following constraints:

- **Boundary:** $u_1 = (1,1)$ and $u_k = (m,n)$. This condition requires the warping path to start and finish in diagonally opposite corner elements of the matrix.
- **Continuity:** Given $u_i = (a,b)$, then $u_{i-1} = (a',b')$, where $a - a' \leq 1$ and $b - b' \leq 1$. This condition restricts the allowable steps in the warping path to connected elements.
- **Monotonicity:** Given $u_i = (a,b)$, then $u_{i-1} = (a',b')$, where $a - a' \geq 0$ and $b - b' \geq 0$. This condition forces the elements of the alignment U to be monotonic in time.

DTW may lead to unintuitive alignments where a single element of one time series maps onto a large subsection of another time series. This undesirable behavior is called a *singularity*. Also unlike the Euclidean distance, DTW is *not* a metric and does not satisfy the triangle inequality. Because of this, many spatial indexing techniques cannot be applied to accelerate it.

Calculating the DTW distance is also expensive and requires $O(mn)$ computational time. To reduce this time, a warping path constraint can be used to limit how far the warping path may stray from the diagonal. Two well-known warping path constraints are the Sakoe-Chiba band and the Itakura Parallelogram [RJ93, SC78]. A Sakoe-Chiba band allows an element x_i to align only with the elements $y_{i-w/2}, \ldots, y_{i+w/2}$ (and vice versa for each y_i), where w is the size of the constraint window. The constraint window can reduce the computational time of DTW to $O(w \cdot \max\{m, n\})$. Since w is a constant value, the computational complexity of DTW is reduced from quadratic to linear.

One final problem with DTW is that it assumes time series can be "stretched" locally by duplicating samples. This results in gaps being filled with replicated values and permits the formation of artificial plateaus. If the time series is generated by sampling some underlying continuous signal, it would be more natural to interpolate values to fill gaps.

2.2.3 Longest Common Subsequence

The *longest common subsequence (LCS)* is the longest sequence that is a subsequence of all sequences in a set of sequences. The LCS method matches two sequences by allowing them to stretch, without rearranging the sequence of the elements but by allowing some elements to be unmatched.

Definition 2.16 *Given two sequences X and Y of the length m and n respectively, the LCS score between X and Y is:*

$$\mathrm{LCS}(X,Y) = \begin{cases} 0 & \text{if } m = 0, n = 0 \\ \mathrm{LCS}(X[2:m], Y[2:n]) + 1 & \text{if } d(x_1, y_1) \leq \epsilon \\ \max\{\mathrm{LCS}(X[2:m], Y), \mathrm{LCS}(X, Y[2:n])\} & \text{otherwise,} \end{cases} \quad (2.12)$$

where ϵ is a threshold value to determine whether or not two elements match. In Equation 2.12, scores are used (rather than distance functions) to compare the similarity.

The higher the score, the more similar the two time series.

The Euclidean distance metric does not allow stretching of time series. The DTW distance measure allows stretching but must match all elements, even the noise. Because of this, both the Euclidean distance and DTW are sensitive to noise. The Euclidean distance may also completely fail in the presence of noise and return a large distance, even though this distance may be caused by only a few pairs of elements of the two sequences. DTW in turn may produce singularities and may generate alignments based on meaningless warping paths.

LCS can handle noise, because the matching threshold uses discrete values 1 and 0 to quantize the distance between two elements. This strategy removes the larger distance effects caused by noise. However, LCS does not differentiate time series with similar subsequences but various gap differences between similar subsequences, which may lead to inaccuracies. Here the gap differences refer to subsequences between two identified similar components of two time series. Two time series may have exactly the same LCS distance to the query sequences, but quite different sizes of gap between the similar subsequences. Like DTW, LCS is not a metric and does not satisfy the triangle inequality. The computational complexity of LCS is quadratic, but as with DTW warping path constraints can also be applied to LCS to reduce the computational complexity.

2.2.4 Dynamic Programming for Sequence Alignment

Sequence alignment methods measure the similarity between two sequences. A sequence alignment method usually consists of an objective function that assigns a score to each possible alignment of sequences. The alignment method produces a pairing of elements from one sequence with the other sequence. The alignment method typically either optimizes an objective function heuristically or guarantees an optimal score such as through a dynamic programming technique. A scoring function is used to accumulate the influence of two matching elements, two mismatching elements, and penalties

for gaps. In pairing elements, gaps can be inserted at any position in the sequences, but the order of elements in each sequence must be preserved. Sequence alignment methods attempt to maximize the alignment score by placing gaps (that are seen as insertion/deletion evolutionary events) in either sequence so as to maximize the number of matching elements and minimize mismatches and gaps.

There are two types of alignment: *global* and *local*. A global alignment attempts to align sequences over their entire length, while a local alignment constructs the best alignment of segments of the sequences that exhibit a high density of matches, ignoring the remaining regions of the sequences. A *pairwise alignment* is an alignment of two sequences and if there are more than two sequences the alignment is called *multiple sequence alignment*.

Sequence alignment problems have been researched well in the area of bioinformatics. Most sequence alignment methods use scoring matrices. Examples of such methods include PAM, BLOSSUM, GONNET, BLAST, and DNA Identity Matrix [AGM$^+$90, Alt91, DSO78, GCB92, HH92]. It is assumed that the matching sequences should have an evolutionary ancestral sequence in common with the query sequence. The alignment path should be the one that requires the fewest evolutionary events. All substitutions are not equally likely and are weighted to account for this. In particular, in genomics insertions and deletions are less likely than substitutions and are weighted appropriately. The choice of scoring matrix determines both the pattern and the extent of substitutions in the sequences which the search is most likely to discover.

Dynamic programming algorithms were initially developed to calculate the minimal edit distance between two sequences [NW70, Sel74]. The first dynamic programming algorithm to compute the edit distance and to search for a pattern sequence within a text was developed by Sellers in 1980 [Sel80]. Many variations have been rediscovered and both theoretical and practical improvements have since been made [CL94, GP90, Ukk85].

Dynamic programming algorithms are particularly flexible in handling different

distance functions, although they are not the most efficient algorithms in general. Dynamic programming routines guarantee the mathematically optimal alignment, and can easily be generalized to optimally align k sequences. However, they take in $O(n^k)$ time, where n is the length of the longest sequence, and hence are unsuitable for when the number of sequences k is large.

Two dynamic programming methods, the *Needleman-Wunsch algorithm* [NW70], which is a global alignment algorithm, and the *Smith-Waterman algorithm* [SW81], which is a local alignment algorithm, will be introduced in the following sections.

Needleman-Wunsch Algorithm

Given two sequences X and Y of the length m and n respectively, the Needleman-Wunsch (NW) algorithm computes the similarity $H(i, j)$ of two sequences ending at position i and j, where $x_i \in X$ and $y_j \in Y$. The computation of $H(i, j)$, for $1 \leq i \leq m$, $1 \leq j \leq n$, is given by the following recurrences:

$$H(i,j) = \max \begin{cases} H(i-1, j-1) + \mathsf{sbt}(i,j) \\ H(i-1, j) - \alpha \\ H(i, j-1) - \beta, \end{cases} \tag{2.13}$$

where sbt is an element substitution cost table. Initialization of these values are given by $H(i, 1) = \mathsf{sbt}(i, 1)$ and $H(1, j) = \mathsf{sbt}(1, j)$. Multiple gap costs are taken into account by α and β, which are the gap penalties. The simplest gap penalties can be constants, or have a form like $\alpha = o + \ell e$, where ℓ is the length of gap, o is the gap "opening" penalty, and e is the "extension" penalty paid per gap position. The value o should be much larger than the value e. Some applications also use a linear gap penalty, i.e., $\alpha = \beta$.

After the scoring matrix has been built, a trace-back procedure is used to find the optimal alignment path, if it is required. This procedure starts the corner with the maximum value, and always selects the maximum value from the outer-most column

and row, and jumps to next maximum in the next row or column.

Smith-Waterman Algorithm

Unlike Needleman-Wunsch algorithm, which looks at each sequence in its entirety, the Smith-Waterman (SW) algorithm compares segments of all possible lengths and chooses whichever optimizes the similarity measure. SW therefore computes a local alignment.

Given two sequences X and Y of the length m and n, respectively, SW computes the similarity $H(i,j)$ of two sequences ending at position i and j, where $x_i \in X$ and $y_j \in Y$. The computation of $H(i,j)$, for $1 \leq i \leq m$, $1 \leq j \leq n$, is given by the following recurrences:

$$\begin{aligned} H(i,j) &= \max\{0, E(i,j), F(i,j), H(i-1,j-1) + \mathsf{sbt}(i,j)\}, \\ E(i,j) &= \max\{H(i,j-1) - \alpha, E(i,j-1) - \beta\}, \\ F(i,j) &= \max\{H(i-1,j) - \alpha, F(i-1,j) - \beta\}; \end{aligned} \tag{2.14}$$

where sbt is an element substitution cost table. Initialization of these values are given by $H(i,1) = \mathsf{sbt}(i,1)$ and $H(1,j) = \mathsf{sbt}(1,j)$. The values α and β are the gap penalties.

The trace-back procedure starts from the element having the highest score. The alignment path is generated as follows: If the current position is the element (i,j), then the next position is $\max\{H(i,j), H(i-1,j), H(i,j-1)\}$, until it reaches zero.

2.2.5 Comparison of Distance Functions

The following criteria are used to evaluate distance functions: (1) ability to handle local time shifting, (2) ability to handle noise, (3) ability to handle different lengths, (4) computation efficiency, (5) whether the distance function is a metric (whether it satisfies the triangle inequality), (6) and what data types they work for, and (7) whether they are whole matching methods or subseries matching methods. Table 2.2 compares

the introduced distance functions according to the above criteria.

Table 2.2: Comparisons of distance functions. The values m and n are the lengths of two time series. Without loss of generality, we assume $m \leq n$.

	Euclidean	**Shift-and-Compare**	**DTW**	**LCS**	**NW**	**SW**
Local Time Shifting	No	No	Yes	Yes	Yes	Yes
Noise Sensitivity	Yes	Yes	Yes	No	No	No
Length Sensitivity	Yes	Yes	No	No	No	No
Time Complexity	$O(n)$	$O(n(n-m))$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
Is a Metric	Yes	Yes	No	No	No	No
Data Type	Time Series	Time Series	Time Series	Discrete Sequence	Discrete Sequence	Discrete Sequence
Matching Type	Whole	Subseries	Whole	Subsequence	Whole	Subsequence

2.3 Dimension Reduction Representations

Transforming the original time series to another (often approximate) representation or segmenting the original data is a common method for both retrieval and compression. Simpler, functional, or lower dimensional representations approximate time series by omitting unimportant details. In this section, dimension reduction methods for time series are introduced.

2.3.1 Transform Methods

Transform methods are based on a change of basis, with the basis chosen to localize a signal in frequency and/or space in order to be able to derive useful properties.

Fourier Transform

The *Fourier transform* is used to transform a function between the time domain and the frequency domain. We assume a signal $g(t)$ is a function that varies with respect to time t. Its corresponding transform in the frequency domain is $G(f)$ where f stands for the frequency of the signal. The Fourier transform $G(t)$ of a continuous signal $g(t)$ is given by

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-i2\pi ft} dt. \tag{2.15}$$

The inverse Fourier transform is given by

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{i2\pi ft} df, \tag{2.16}$$

where $i = \sqrt{-1}$.

The *discrete Fourier transform (DFT)* is a dimension reduction method that has been used to index time series data [AFS93]. DFT describes a time series by a set of *sampled* sine/cosine waves. A time series of length n is decomposed into n complex sinusoids that can be combined to reconstruct the original data. DFT coefficients can be used to approximate the original time series by eliminating coefficients that have low amplitudes.

Given a time series $g = (g_0, \ldots, g_{n-1})$, its DFT is a sequence $G = (G_0, \ldots, G_{n-1})$,

$$G_j = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} g_i e^{-i2\pi ij/n}. \tag{2.17}$$

The *inverse discrete Fourier transform (IDFT)* is

$$g_i = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} G_j e^{-i2\pi ij/n}, \tag{2.18}$$

where $i, j \in \{0, \ldots, n-1\}$.

According to Parseval's Theorem [OS75], the L_p-norm distance metrics on the first few DFT coefficients are lower bounds of the Euclidean distance in the original data space [FRM94]. Therefore, an index that is created using the first few DFT coefficients can be designed so that it ensures no false dismissals.

Wavelet Transform

The Fourier transform does not specify where the function $f(t)$ (i.e., for what value of t) has a certain frequency. *Wavelet transforms* can solve this problem using a set of basis functions that are localized in both time and space. Wavelet transforms select a mother wavelet that is nonzero in some small interval, and use it to analyze the properties of $f(t)$ in that interval. Then the mother wavelet is translated to another interval of t to analyze the properties of that interval. The mother wavelet $\psi(t)$ must satisfy

$$\int_{-\infty}^{\infty} \psi(t)dt = 0, \tag{2.19}$$

$$\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty. \tag{2.20}$$

Once the mother wavelet has been chosen, the *continuous wavelet transform (CWT)* of a signal $f(t)$ is defined as

$$W(\tau, s) = \int_{-\infty}^{\infty} f(t)\psi_{\tau,s}(t)dt, \tag{2.21}$$

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right). \tag{2.22}$$

The parameter s is a translation parameter and τ is a scaling parameter.

The *discrete wavelet transform (DWT)* is based on sub-band coding and yields a fast computation for a wavelet transform. In CWT, the signal is analyzed using a set of

related basis functions by scaling and translation. In DWT, a time-scale representation of a digital signal is obtained using a set of discrete filters. The signal passes through filters with different cutoff frequencies at different scales.

DWT has been used as a dimension reduction method in previous time series indexing work [CF99, PM01, WAA00]. DWT is a multi-resolution representation of time series and approximates the time series from global sequences to local subseries. Unlike DFT which only offers frequency information, DWT offers time-frequency location information as well.

Compared to DFT, DWT describes a time series both at various locations and at various time granularities. Each time granularity refers to the level of detail that can be captured by the DWT. To measure the similarity of two time series, the Euclidean distance can be used over transformed DWT coefficients [PM01, CF99]. Also, there is not just one DWT but many, depending on the choice of basis. Different basis can have different properties.

The *Haar wavelet* is a simple wavelet proposed by Alfréd Haar in 1910 [Chu92]. Any signal $f(t) \in L_2$-norm space can be uniquely represented by the following series, which is called the *Haar wavelet transform (HWT)* of function $f(t)$,

$$f(t) = \sum_{\tau=-\infty}^{\infty} a_\tau \phi(t-\tau) + \sum_{\tau=-\infty}^{\infty} \sum_{i=0}^{\infty} b_{i,\tau} \psi(2^i t - \tau), \tag{2.23}$$

where a_τ and $b_{i,\tau}$ are coefficients to be calculated. The basic scale function $\phi(t)$ is the unit pulse:

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.24}$$

The basic Haar wavelet $\psi(t)$ is the following piecewise constant function:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.25}$$

2.3.2 Principal Component Analysis

As a dimension reduction method, *principal component analysis (PCA)* has been widely used in content-based retrieval of texts [HLB94], time series [KJF97], and images [KAS98, PC03]. PCA is based on the fact that given a set of high-dimensional points, variations between points tend to be concentrated along a small number of axes, on which they can be discriminated from each other. PCA is used to find these axes and determines a transformation to map the points to a low-dimensional basis.

A set of m-dimensional data points can be represented as an $n \times m$ matrix A, where n is the number of data points in the set. PCA first extracts the empirical mean of the distribution. The mean value is calculated and subtracted for each dimension:

$$B(m) = \frac{1}{n} \sum_{i=1}^{n} A(i, m) \tag{2.26}$$

$$\bar{A} = A - B. \tag{2.27}$$

The matrix A can be factored using an *singular value decomposition (SVD)*:

$$\bar{A} = U \Sigma V^T, \tag{2.28}$$

where U is a column-orthogonal $n \times m$ matrix, Σ is a diagonal $m \times m$ matrix of the eigenvalues of the matrix $\bar{A}$, and V is a column-orthogonal $m \times m$ matrix. The matrices U and V are orthogonal.

The PCA transformation is given by $U \times \Sigma$, which can be computed by $\bar{A} \times V$. The matrix $U \times \Sigma$ is a rotation and scale of the matrix $\bar{A}$. The number of dimensions of the

transformed data does not change. However, by taking only the first k ($k \leq m$) largest eigenvalues of Σ and corresponding entries in $\bar{A}$, U, and V, the size of dimensions can be reduced from m to k in the transformed space. However, the transformation ensures that most of the variation is concentrated in these initial dimensions.

In terms of minimizing the reconstruction error, PCA is better than DFT and DWT. However, PCA is expensive, and needs $O(nm^2)$ computational time and $O(nm)$ space. PCA also needs the whole database to compute the transformation[2], which assumes the database is not updated frequently. PCA also assumes the query follows the same data distribution as the database. If this assumption is not satisfied, PCA may introduce a loose lower bound of the distance between the data, which may lead to many false positives.

2.3.3 Piecewise Approximations

Piecewise approximation methods divide the time series into segments and approximate each segment using functions. In this section, various piecewise approximation methods are surveyed.

Piecewise linear approximation (PLA) has been widely used in time series analysis [Pav73] and retrieval [SZ96]. This method divides the time series into segment and approximates each segment using a linear function that is desirable to minimize error.

Morinaka et al. [MYAU01] proposed a modified L_p-norm distance over the PLA representation to quickly find the approximate matches without false dismissals. The modified L_p-norm distance is corrected by the value of potential approximation error deviation of a segment, resulting in a bound on the true distance. The computational complexity of this distance function is linear. However, it is hard to build the index for this distance function.

Yi and Faloutsos [YF00] and Keogh et al. [KP00] proposed a method called *piece-*

[2]The database could be sub-sampled randomly to compute the transformation on a smaller data set, but this sample has to be representative.

wise aggregate approximation (PAA). PAA divides a time series of length n into k segments of equal length and approximates each segment using the average value of the segment. In this way, a n-dimensional point (the sequence of n samples) is mapped to a k-dimensional point (the sequence of k segment averages) where $k < n$. Considering a time series $X = (x_1, \ldots, x_n)$, its piecewise constant representation is $S_X = (s_1^X, \ldots, s_k^X)$, where

$$s_i^X = \frac{k}{n} \sum_{j=\frac{n}{k}(i-1)+1}^{\frac{n}{k}i} x_j. \tag{2.29}$$

The distance between PAA representations S_X and S_Y of time series X and Y in the reduced dimension can use the general L_p-norm metric, weighted L_p-norm metrics, and Euclidean distance metrics, and these can be used as approximations to the true distances between the original time series.

Uniform segmentation makes it difficult to perform approximation error control. Morinaka et al. [MYAU01] used the *least square approximation* method to segment a time series to lines. Each line is the longest possible segment whose accumulated error does not exceed a predefined tolerance. An approximate distance is defined over two segment sequences based on the linear functions that describe each segment. Two different distances are defined, including one to deal with the case when line segments intersect each other, and the other when they do not.

Because the approximate distance of two segment sequences may be larger than the real distance between the original time series, a bound called *worst error deviation (WED)*, is used to bound the approximate error deviation of a segment. For each line segment, its WED is computed based on difference between the values of original data that are above and below the segment. By adding the WED to the approximate distance, it is possible to guarantee that modified approximate distance is the lower bound of the real distance in the original time series space.

Keogh and Smyth [KS97] proposed a probabilistic method which represents each segment by a set of local features (such as peaks, troughs, and plateaus). These features

are extracted from the time series to describe the properties of the original data. The overall distance is computed based on the local features and relative positions of individual features. Unfortunately, the computational complexity of this distance function is exponential. Even when combined with some heuristics [KS97], the computational cost is quadratic.

Adaptive piecewise constant approximation (APCA) proposed in [KCMP01] is another non-uniform segmentation and approximation method. The intuition of this approximation is that regions with large fluctuations are represented with short segments, and flat regions are represented with long segments. The APCA method converts the problem into a wavelet compression problem to find a solution. Then it converts the solution back to a PLA representation. Given a time series $X = (x_1, \ldots, x_n)$, its APCA representation is

$$S'_X = ((xm_1, xr_1), \ldots, (xm_k, xr_k)), \tag{2.30}$$

where xm_i is the mean value of the data points in i^{th} segment, xr_i is the right endpoint of i^{th} segment, and k is the number of segments.

Keogh et al. [KCMP01] proposed a distance function on the APCA representation to compute a lower bound on the Euclidean distance. Given two time series X and Y, their corresponding APCA representations S'_X and S'_Y are obtained by projecting the endpoints of S'_X onto X and computing the mean value of the sections of X that fall within the projected interval. That is,

$$S'_X = ((qm_1, qr_1), \ldots, (qm_k, qr_k)) \tag{2.31}$$

where

$$\begin{aligned} qr_i &= xr_i, \\ qm_i &= \frac{1}{r_i - r_{i-1} - 1} \sum_{j=r_{i-1}+1}^{r_i} x_j. \end{aligned}$$

The lower bounding distance is defined as

$$d(S'_X, S'_Y) = \sqrt{\sum_{i=1}^{k} (xr_i - xr_{i-1})(qm_i - xm_i)^2}. \tag{2.32}$$

Keogh et al.'s experimental results have demonstrated that APCA outperforms other indexable representations such as DFT and DWT in terms of efficiency.

2.3.4 Symbolic Representation

Since many distance functions, algorithms, and data structures have been developed for indexing strings, it is intuitive to consider the possibility of converting real-valued time series data into symbolic, string-like representations and applying string matching techniques to time series retrieval. Agrawal et al. [APWZ95] proposed *SDL* that is a language for describing and retrieving the "shape" of one dimensional time series. The "shape" is defined based on the difference of every pair of consecutive values, which is quantized and represented by a distinct symbol of a predefined alphabet.

Huang and Yu [HY99] proposed an *interactive matching of patterns with advanced constraints in time-series (IMPACT)* method, which converts a time series to a string by changing ratios between consecutive values. A general suffix tree is used to index the strings. IMPACT can handle dynamic query constraints with different degrees of accuracy and dynamically specified combinational patterns. Lin et al. [LKLP02] proposed a symbolic representation of one dimensional time series, called *SAX*, by first transforming it to a piecewise approximation. Then, the values of the piecewise approximation are quantized and each is mapped to a symbol. Instead of mapping values, average values, and differences between values of time series data to symbols, some other methods convert movement slopes of time series data into symbols using best-line fitting algorithms [QWW98, SZ96].

The distance functions for symbolic representations are either exact symbol equal-

ity matching functions [RJ93, APWZ95, SZ96, HY99] or modified Euclidean distance metrics [LKLP02].

The methods that convert time series data into strings offer opportunities to apply text indexing methods for similarity search over time series. Andŕe-Jönsson and Badal [AJB97] proposed a method that used signature files to represent time series data. First, the time series is converted into strings by quantizing amplitude differences into discrete symbols from a predefined alphabet [APWZ95]. Then, signature files are generated by sliding a window along each string and mapping the text in the window into a number of signature bits. Similarly, a query signature is generated from the query. Finally, a linear scan is carried out to search signature files in the database. The advantage of this method is that a signature file is compact and searching signature files is linear time. Because essentially this method is a hash table approach, searching for each symbol requires constant time.

Indexing using signatures can ensure no false dismissals [LKLP02]. However, since signatures cannot avoid false positives, the obtained results need to be verified by conducting a similarity search on the original time series data. Because of the loss in accuracy during the conversion from a real value to a symbol, many false positives may be introduced.

2.4 Indexing Frameworks

For large datasets, indexing is necessary for efficient search and other data mining tasks. The GEMINI framework, which is widely applied in indexing time series, is introduced in this section. Several dominating lower bound strategies, which guarantee no false dismissals during index-based search, are also reviewed.

2.4.1 GEMINI Framework

The GEMINI framework [FRM94] has been widely used for indexing and retrieval in a wide range of applications, but specifically in time series indexing. By framework, we should note that this is a general approach used by a wide variety of more specific methods. As shown in Figure 2.2, an indexing method following the GEMINI framework builds an index by first transforming the data to a lower-dimensional representation. A time series of length n is transformed to a set of k-dimensional points, where $k < n$. A bounding distance function needs to be defined on the transformed representation to guarantee no false dismissals. These k-dimensional points can be indexed in the spatial access method, such as an R-tree, using this bounding distance function. Given a query of length m, it likewise is transformed into a k-dimensional point using the same dimension reduction method. Each low-dimensional query point searches for its best match in the index. If the query time series is segmented and mapped to more than one lower-dimensional query point, for each query point, a matching point is found and a post-processing procedure will connect all the matching points.

Basically there exist two kinds of dimension reduction techniques: exact and approximate. Exact dimension reduction techniques guarantee that no false dismissals (false negative) occur when queries are executed on the reduced dimensional space, although it might return false positives. To achieve this, the distance function defined in the reduced dimensional space must be a lower bound of the "true" distance in the original space. Approximate dimension reduction techniques do not guarantee this, therefore, no lower bounding distance function needs to be defined. Since the "no false dismissals" guarantees correctness of the retrieval, most methods use the exact dimension reduction approach. False positives can be culled in a post-processing step, although of course a large number of false positives will lead to inefficient search.

The simplest way to apply the GEMINI framework to time series indexing is to slide a window of size w at every possible offset of the time series of length n. For each such placement of the window, the segment in the window can be approxi-

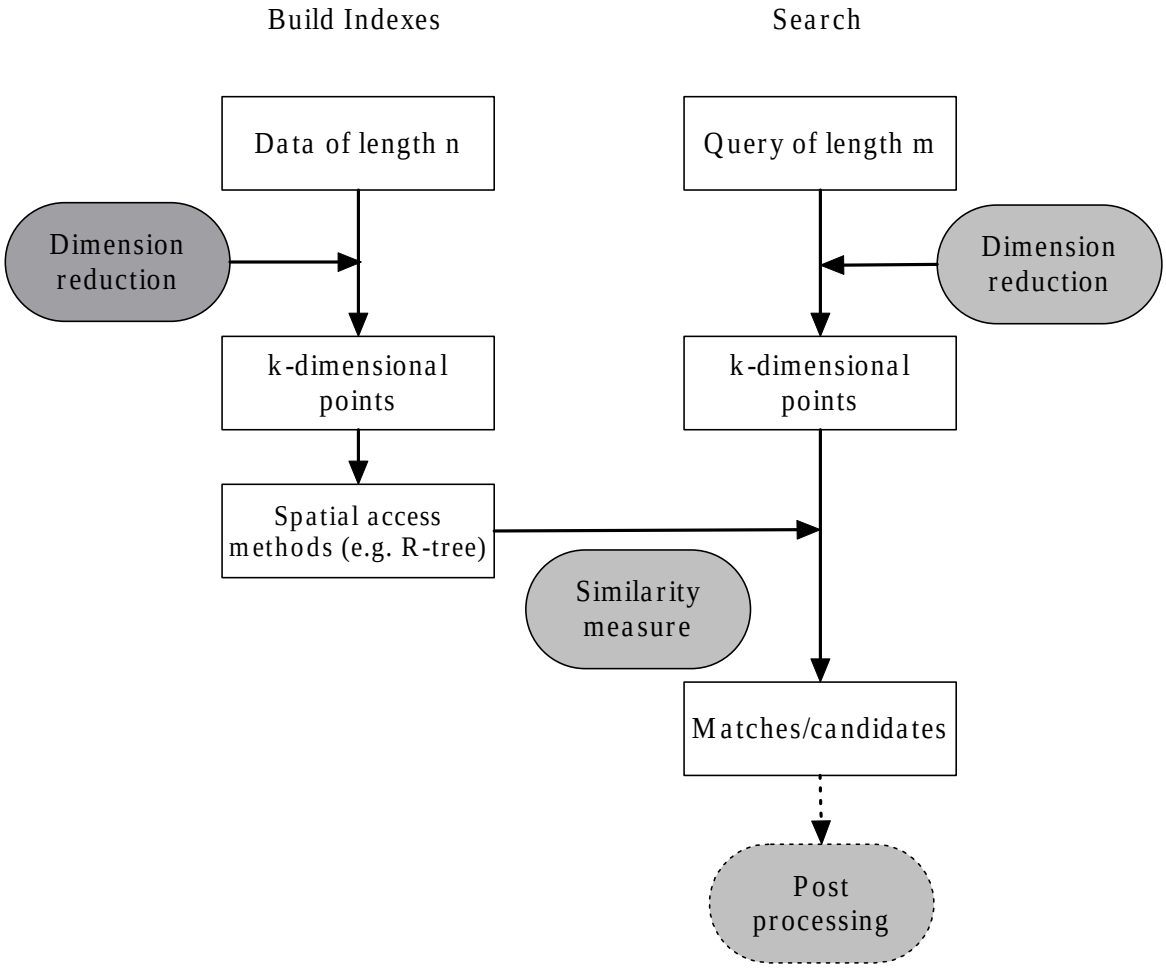


Figure 2.2: Overview of the GEMINI framework.

mated in some way using k parameters. Several representations have been used with just such an approach, including those already surveyed: Fourier transforms [AFS93], wavelet transforms [CF99], average values in adjacent windows [KCMP01], and bounding boxes [VHGK03]. In this way, the original time series is mapped to a sequence in feature space consisting of $k = n - w + 1$ points. A distance metric is defined over this approximation that underestimates the true distance between the time series. The approximations can be indexed by a spatial data structure, such as an R-tree [Gut84], R*-tree [BKSS90], MVP-tree [BO99], M-tree [CPZ97], or SA-tree [Nav02].

The GEMINI framework aims to localize the search to a portion of the database that is close to the query. The low-dimensional approximation makes coherent spatial access methods viable for indexing the database efficiently [BBK01]. However, when the lengths of the input segments increase, the storage requirement increases linearly and the performance suffers while the R-tree depth increases.

Unfortunately, the sliding window approach is redundant. One improvement is to divide the feature sequence to subseries and to represent each subseries with its minimum bounding envelope. The subdivision criterion should try to minimize the number of disk access. One straightforward method is a uniform subdivision according to a pre-determined size of subseries. Another method is to use a function of lengths for the subseries size.

To search a database, the query is first mapped to a feature sequence in the same feature space. Then similar candidate matches are found in the hierarchical spatial index structure. Finally the query is compared with the candidate matches in the original space. If the length of the query is larger than w, the query is chopped into subseries of length w. Each subquery is then processed and the candidates are merged before the final match.

The most common approach to exact dimension reduction uses the sliding window, i.e., indexing all possible subseries of given length w. A sliding window with fixed size w is commonly used. There are three main algorithms in this category: FRM [FRM94],

DualMatch [MWL01], and J-sliding windows [MWH02]. FRM divides data into sliding windows of size w and transforms each window to a lower-dimensional point. It then divides the query Q into $\lfloor Q\#/w \rfloor$ disjoint windows of size w. However, as noted earlier, the search performance may become poor if the R-tree is tall. To solve this problem, FRM stores only the minimum bounding envelopes.

DualMatch takes the opposite approach to FRM. It first divides data into $\lfloor X\#/w \rfloor$ segments and slides a window over the query. DualMatch improves performance significantly. However, DualMatch has the problem of having a smaller allowable window size—approximately half that of FRM.

J-sliding windows compromise between FRM and DualMatch by dividing data into generalized sliding windows (J-sliding windows) and the query into generalized disjoint windows (J-disjoint windows). "J-sliding" means the sliding offset is not 1 but J, where $1 \leq J \leq w$. Various distance functions, such as Euclidean distance [FRM94, MWH02], DTW [WW03], and LCS [HKK07], can be used. These algorithms produce a set of segments of equal length w.

The GEMINI framework uses a lower bound in a lower dimensional space of the true distance in the original space to guarantee no false dismissals when the index is used as a filter. A lower bounding function should have the following properties.

- **Accuracy**: It must return all the qualifying subseries as candidates. It may produce false positives, which can however be discarded after a more expensive exact comparison. That means the distance between each of the transformed time series should be less than the distance between any pair of original time series X and Y, i.e.,

  $$d_{lb}(X, Y) \leq d(X, Y), \tag{2.33}$$

  where d_{lb} is the lower bounding distance function and d is the original distance function.

- **Efficiency**: The time complexity for computing the lower bound distance should

be low. Also, to implement R-tree search, it should be possible to bound the minimum distance between bounding volumes in lower-dimensional space. Furthermore, if the lower bounding function satisfies the triangle inequality, it can be used as a filtering function in indexing search.

- **Tightness**: The lower bound should be as tight as possible to avoid an excessive number of false positives. For example, 0 is a lower bound but using this trivial bound for all distance functions would return the entire database as a candidate set for every query.

2.4.2 Lower Bounding Methods

There exist three dominant lower bounding methods on the Euclidean distance metric and Dynamic Time Warping (DTW). All these methods have been proven to guarantee no false dismissal relative to the Euclidean distance and DTW and to satisfy the triangle inequality.

Kim et al.'s method [KPC01] extracts a 4-tuple vector,

$$S_X = (\text{First}(X), \text{Last}(X), \text{Greatest}(X), \text{Smallest}(X)),$$

from each time series X. The features are the first, last, greatest and smallest elements of X, respectively. The lower bounding function d_{lb-Kim} for two time series X and Y is defined as

$$d_{lb-Kim}(X, Q) = \max \begin{cases} |\text{First}(X) - \text{First}(Y)| \\ |\text{Last}(X) - \text{Last}(Y)| \\ |\text{Greatest}(X) - \text{Greatest}(Y)| \\ |\text{Smallest}(X) - \text{Smallest}(Y)|. \end{cases} \tag{2.34}$$

The lower bounding method introduced by Yi et al. [YJF98] takes advantage of the observation that all the points in one sequence that are larger (or smaller) than the

maximum (or minimum) of the other sequence must contribute at least the squared difference of their value and the maximum (or minimum) value of the other sequence to the final DTW distance.

Yi's method [YJF98] considers three possible arrangements of the ranges, $R_X = [\max(X), \min(X)]$ and $R_Y = [\max(Y), \min(Y)]$, of the pair of time series X and Y. Yi et al.'s function is the following:

$$d_{lb-Yi}(X,Y) = \begin{cases} d_1(X,Y) & \text{if } R_X \text{ and } R_Y \text{ overlap assuming } \max(X) \geq \max(Y) \\ d_2(X,Y) & \text{if } R_X \text{ encloses } R_Y \\ d_3(X,Y) & \text{if } R_X \text{ and } R_Y \text{ are disjoint assuming } \min(X) \geq \max(Y), \end{cases} \tag{2.35}$$

$$d_1(X,Y) = \sum_{x_i > \max(Y)} |x_i - \max(Y)| + \sum_{y_i < \min(X)} |y_i - \min(X)|, \tag{2.36}$$

$$d_2(X,Y) = \sum_{x_i > \max(Y)} |x_i - \max(Y)| + \sum_{x_i < \min(Y)} |x_i - \min(Y)|, \tag{2.37}$$

$$d_3(X,Y) = \max\{\sum_{i=1}^{n} |x_i - \max(Y)|, \sum_{j=1}^{m} |y_j - \min(X)|\}. \tag{2.38}$$

Keogh [Keo02] method considers the global path constraints of DTW, $i - r \leq j \leq i + r$. Keogh's lower bounding method is the following:

$$\begin{aligned} U_i &= \max(Q[i-r : i+r]), \\ L_i &= \min(Q[i-r : i+r]), \\ d_{lb-keogh} &= \sqrt{\sum_{i=1}^{n} \begin{cases} (x_i - U_i)^2 & \text{if } x_i > U_i \\ (x_i - L_i)^2 & \text{if } x_i < U_i \\ 0 & \text{otherwise} \end{cases}}. \end{aligned} \tag{2.39}$$

2.5 Summary

In this chapter, the definitions and underlying frameworks relevant to time series retrieval and compression, including similarity measures, dimension reduction represen-

tations, and indexing techniques, were introduced. The properties and performance of different methods were compared. Generally, different representations require different similarity measures and result in different strategies, although there are some common themes. Most indexing techniques require the similarity measure to be a metric, i.e., the similarity measure should satisfy the triangle inequality. However, common similarity measures for subseries matching do not satisfy this requirement.

In the following chapters, a representation, a similarity measure, an indexing and search method for subseries join of time series will be proposed. The representation is based on a scale-space analysis and a non-uniform segmentation method. Compared to previous work, the segmentation method is based on the intrinsic structure of the time series. The scale-space analysis generates a hierarchical representation which includes coarse to fine details of time series. Based on this hierarchical representation, indexing and search methods will be proposed to solve the subseries join problem that was defined in this chapter.

Chapter 3

Non-uniform Segmentation and Representation

As introduced in Section 2.4.1, for subseries matching, a sliding window is commonly used to index all possible subseries of a certain size of the window. This framework is based on a uniform segmentation. However, a uniform segmentation requires the user to manually select the length of subseries, and is not sensitive to the actual behavior of the data. This arbitrary segmentation may cause unnecessary division of important features in the data into different subseries. Overlapping sliding windows can avoid division of features but at the cost of a redundant representation.

In this chapter, a non-uniform segmentation method is proposed based on an anisotropic diffusion scale-space analysis. A minimal polynomial envelope is used to approximate each segment, and a feature tree is used to represent a time series. Based on this representation, an indexing scheme and a compression scheme is developed to hierarchically compress storage of features.

3.1 Non-uniform Segmentation

This section shows how to convert the original time series into a feature sequence using scale-space analysis (see Figure 3.1). The scale-space analysis actually gives a hierarchy of non-uniform features but this hierarchy is encoded into a linear sequence using a post-order traversal, as shown in Section 3.2.

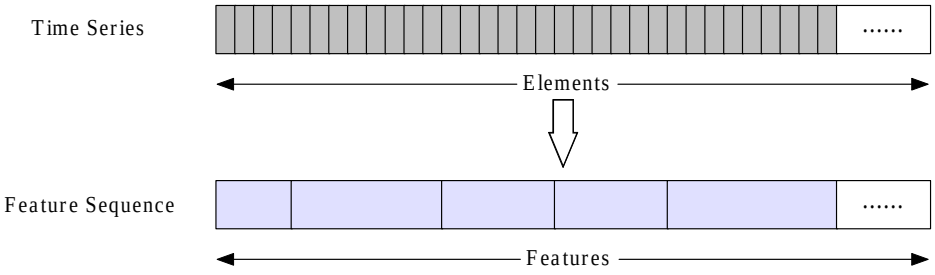


Figure 3.1: Convert a time series into a feature sequence using non-uniform segmentation.

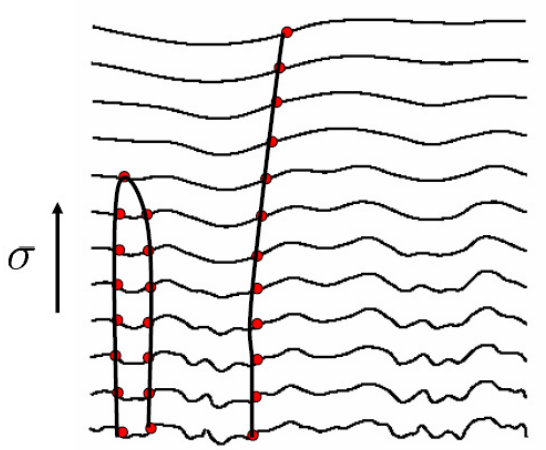
3.1.1 Scale-space Smoothing

One of the first descriptions of scale-space smoothing was in the area of vision, and in particular was applied to the problem of matching stereoscopic images introduced by Marr and Poggio [MP79], and was also applied to the creation of primal sketches of images (Marr and Hildreth [MH80]). Witkin [Wit83] proposed a scale-space method that generated coarser resolution images by convolving the original image with a gradually widening Gaussian kernel.

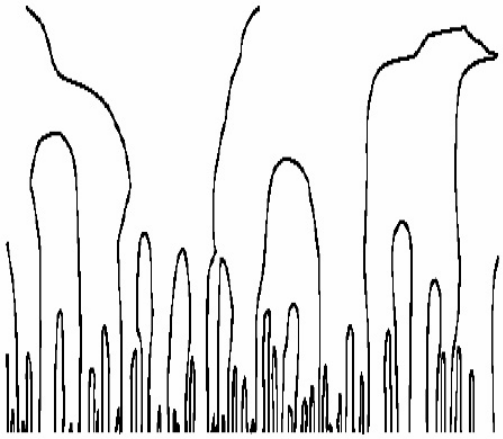
Gaussian convolution of a signal $f(x)$ is given by

$$G(x,\sigma) = f(x) * g(x,\sigma) = \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} f(u) e^{-(x-u)^2/2\sigma^2}. \tag{3.1}$$

By changing the scale parameter σ in a Gaussian kernel, the signal is smoothed at different levels. An example is shown in Figure 3.2(a).



(a) A sequence of Gaussian smoothing in a scale space.



(b) Contours of $G_{xx} = 0$ in a scale space.

Figure 3.2: Gaussian smoothing. (From Figure 1 and Figure 2 in Witkin's paper [Wit83].)

The extrema of slope, i.e., the inflection points, are given by a subset of the zero-crossings of the second derivative;

$$\frac{\partial^2 G}{\partial x^2} = f * \frac{\partial^2 g}{\partial x^2} = 0, \tag{3.2}$$

$$\frac{\partial^3 G}{\partial x^3} \neq 0. \tag{3.3}$$

The contours of these inflection points over increasing scales are shown in Figure 3.2(b). Discontinuities merge at coarser scales and this generates a hierarchy of features. The problem with Gaussian smoothing is that the locations of discontinuities "migrate" with scale so that the fine segmentation is not aligned with the coarse segmentation. Tracking this movement also complicates analysis.

Anisotropic Diffusion

Perona and Malik [PM90] proposed a noise reduction method for images using anisotropic diffusion, which can be seen as a generalization of Gaussian smoothing but without the discontinuity migration problem. The idea of the Perona-Malik method is a filter defined as a modified diffusion process that encourages intra-region smoothing while inhibiting inter-region smoothing. Their smoothing process can avoid the blurring and localization problems of filters based on convolution. Although alternatives are possible, including bilateral filters [TM98], anisotropic diffusion filters are preferred because they automatically generate a scale space that maintains the positions of discontinuities. The term "anisotropic" generally means that the smoothing (diffusion) process is different in different directions. In 2D, this refers to different radial directions around a point. In 1D, it simply means that the smoothing to the left may be different from the smoothing to the right at each point.

Given a continuous signal $X(x)$, the continuous form of the 1D anisotropic diffusion filter is given by the solution to

$$\frac{\partial}{\partial s}X(x,s) = \frac{\partial}{\partial x} \cdot \left(c(x,s)\frac{\partial}{\partial x}X(x,s) \right), \tag{3.4}$$

where s is scale. The function c is a conductance function that returns a value in the range of $[0,1]$. It is a function of the gradient of X and should be a monotonically decreasing function of the gradient's magnitude. One of the following definitions can be chosen:

$$c(x,s) = \exp\left(-\left(\kappa^{-1}\frac{\partial}{\partial x}X(x,s) \right)^2 \right), \tag{3.5}$$

$$c(x,s) = \frac{1}{1 + (\left|\frac{\partial}{\partial x}X(x,s)\right| / \kappa)^2}. \tag{3.6}$$

If we discretize Equation 3.4 using the sequence $X = (x_1, \ldots, x_n)$ and replace the

scale s with the number of iterations $\sigma = \lambda^{-1}s$, we get the following implementation:

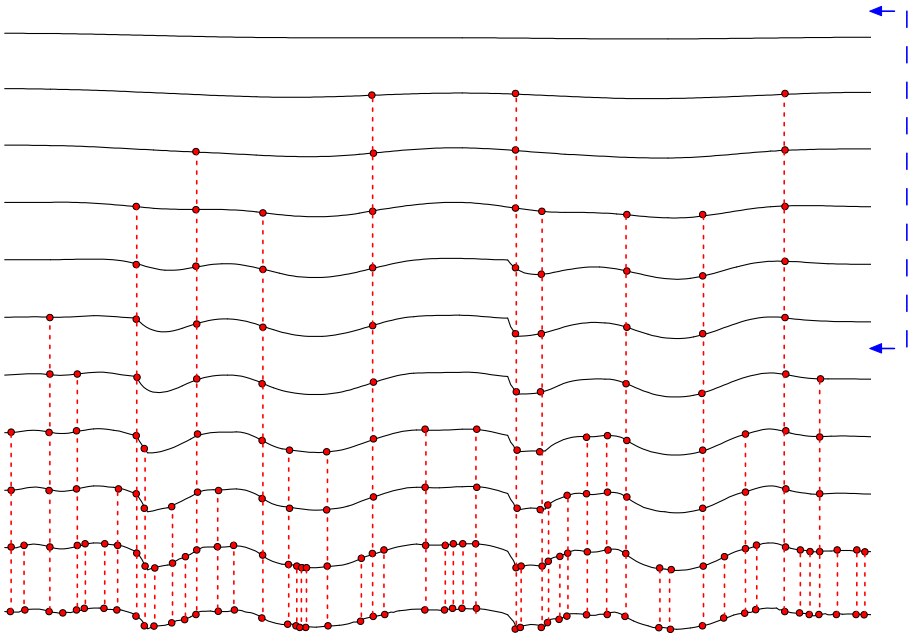
$$x_i^{\sigma+1} = x_i^{\sigma} + \lambda c(i+1,\sigma)(x_{i+1}^{\sigma} - x_i^{\sigma}) - \lambda c(i-1,\sigma)(x_i^{\sigma} - x_{i-1}^{\sigma}). \tag{3.7}$$

The boundary conditions are $x_i^0 = x_i$, $x_1^{\sigma} = x_1$, and $x_n^{\sigma} = x_n$. For stability, we must have $0 \leq \lambda \leq 1/4$. The function c is called the *conductance function*. The conductance values are conceptually interdigitated with the smoothed signal with c_i between x_i and x_{i+1}. It should be computed using $c(i,\sigma) = g(x_{i+1}^{\sigma} - x_i^{\sigma})$. The function g takes the form given by Equation 3.5 or Equation 3.6, but with the finite difference given replacing the gradient.

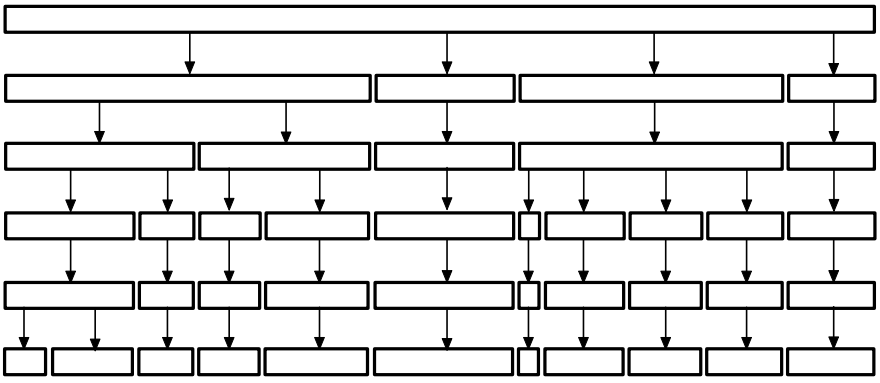
The value κ is referred to as the *diffusion constant* and controls the rate of conduction. If κ is low, small intensity gradients are able to block conduction and hence inhibit diffusion across step edges. A large value, in contract, reduces the influence of intensity gradients on conduction. The constant κ can be selected either by hand or using the "noise estimator" proposed by Canny [Can86]. An overly small value of the constant κ may cause staircases in smoothing and also greatly slows convergence; however, the Canny noise estimator generally avoids this problem. The Canny noise estimator computes a histogram of the absolute values of the gradient and sets κ to the 90% value of its integral at each iteration. However, the Canny noise estimator is relatively slow, especially when the number of iterations is large. Since $|\frac{\partial}{\partial x}X(x,s)| \approx 2\kappa$ is approximately where the value of the conductance function (Equation 3.5) drops to zero, an alternative is to set the value of κ to

$$\kappa = \frac{|\max(X) - \min(X)|}{2}. \tag{3.8}$$

The values $\max(X)$ and $\min(X)$ are the maximum value and minimum values of X, respectively. This value is reset after every iteration. If $\kappa = 0$, which means all values in X are equal, then the anisotropic diffusion process (Given by Equation 3.7) stops. Figure 3.3(a) shows a result from an application of the anisotropic diffusion process.



(a) The boundary points of same positions on different scales are shown. The boundary points are lined up at the same positions in different scales, and in this diagram are aligned by dashed lines.



(b) The hierarchical structure of the scales marked by long dashed lines and arrows at the right side of (a).

Figure 3.3: The scale space generated by anisotropic diffusion and the hierarchical structure generated by tracking the locations of zero crossings of the second derivative across scales.

3.1.2 Segmentation at Discontinuities

Anisotropic diffusion smoothes the curve over a scale space. To extract the boundary points of a series at a particular scale, the 1D Canny edge detector [Can86] is applied to the smoothed curve. The theory of edge detection was introduced by Marr and Hildreth in their early paper [MH80]. The Canny edge detector detects boundaries at the zero-crossings of the second derivative of data and the gradient magnitude is also above some threshold $\epsilon_b > 0$, i.e.,

$$\frac{\partial^2}{\partial x^2} X(x,s) = 0, \tag{3.9}$$

$$\left| \frac{\partial}{\partial x} X(x,s) \right| \geq \epsilon_b. \tag{3.10}$$

The positions of the zero crossings of the second derivative are invariant under anisotropic diffusion and so can be aligned across scales. Coarser scales simply eliminate weaker boundary points (see Figure 3.3(a)).

There is a slight paradox here: smoothing across edges identified by large values of the second derivative is inhibited but edges with zero values of the magnitude of the second derivative are identified as edges by the edge detectors. This paradox can be resolved by realizing that step edges are associated with both a large positive spike and a large negative spike in the second derivative. It should be noted this definition of boundary points also segments the curve into regions of positive and negative acceleration, which is consistent with a categorization of segments into concave and convex regions.

In summary, the anisotropic diffusion process generates a scale-space analysis of a signal and segmentation of this scale space produces a hierarchical representation (Figure 3.3(a)). Moving from fine to coarse, two or more segments may be merged into a single segment at each iteration because of the erosion of boundary points. As shown in Figure 3.3(b), this structure can be represented as a tree.

3.2 Feature Representation

The hierarchical representation generated by scale-space analysis and non-uniform segmentation can now be used to generate an index for each element of the time series dataset. This index can be used to accelerate both subseries matching and subseries join.

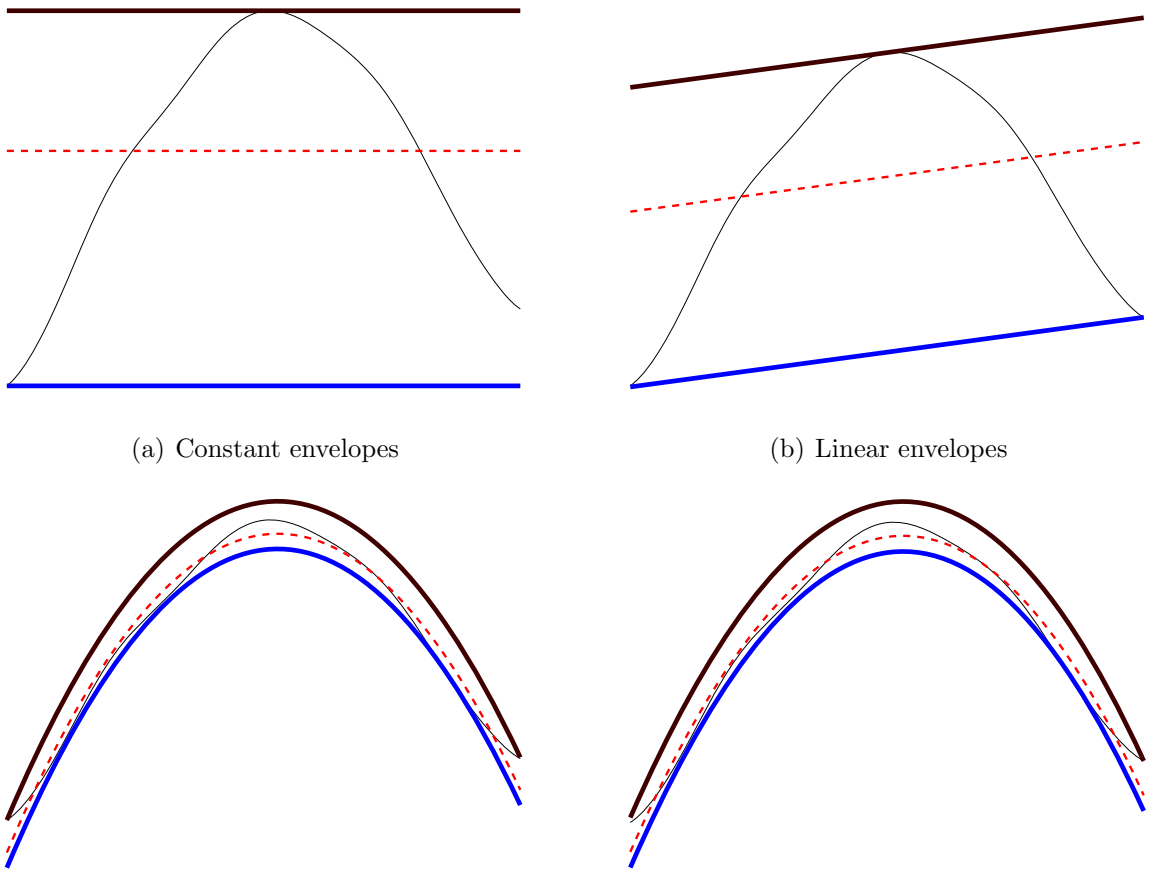
This section describes how each segment is approximated with a feature. This feature has a finite number of parameters and can be used to compute distance bounds between segments.

3.2.1 Minimum Polynomial Envelope

Given a segment $A = X[i : j]$, parameters of length and shape (based on a fitting polynomial) are used to characterize it. Such a representation is called a *feature*. The length parameter gives the number of elements in this segment, specifically, $|A| = j - i + 1$. Inclusive indices for i and j are used, so when interpreted in the continuous domain, this is the same as if the segments are split halfway between samples, consistent with the interpretation of time series as samples of a continuous function.

A polynomial $P(A, t)$ is then used to approximate the shape of each segment A, with t being a real value varying over $[i - 1/2, j + 1/2]$. The linear mapping $t_{i:j} = \tau(j-i+1)+(i+1/2)$ reparameterizes the polynomial over $[0, 1]$. This reparameterized polynomial is represented as $P(A, \tau) = P(A, t_{i:j}(\tau))$. To derive minimal envelopes, the constant part of this polynomial can be replaced with an interval to bound the original fine-scale data. Polynomial approximations and minimal envelopes of different orders are shown in Figure 3.4.

Now we will show how distances between polynomial approximations can be computed and distances between functions enclosed with minimal polynomial envelopes can be bounded. Given two polynomials A and B, the distance between them can be



(a) Constant envelopes

(b) Linear envelopes

(c) Quadratic envelopes

(d) Cubic envelopes

Figure 3.4: The dashed curves are fitting polynomials of various orders. The thick curves are the minimal polynomial envelope (MPE).

defined as

$$d_u^2(A,B) = \int_0^1 (P(A,\tau) - P(B,\tau))^2 + \theta(|A| - |B|)^2 \, d\tau, \tag{3.11}$$

$$d_u(A,B) = \sqrt{d_u^2(A,B)}, \tag{3.12}$$

where $P(A,\tau)$ and $P(B,\tau)$ are the rescaled polynomials fitting segments A and B respectively (representing the shape) and θ is a weight that penalizes the difference in lengths.

This distance can be computed analytically from the coefficients of the polynomials and the lengths of the segments. In fact, the polynomial coefficients and the segment lengths can be placed in a single vector and mapped through a linear transformation so that ordinary Euclidean distances on the transformed coefficients correspond exactly to d_u as defined above.

Consider the specific case of quadratic polynomials. Then the two rescaled polynomials are given by

$$\begin{aligned} P(A,\tau) &= a_2\tau^2 + a_1\tau + a_0, \\ P(B,\tau) &= b_2\tau^2 + b_1\tau + b_0. \end{aligned}$$

Defining $c_i = a_i - b_i$ the difference $P(A,\tau) - P(B,\tau)$ is given by

$$P(A,\tau) - P(B,\tau) = c_2\tau^2 + c_1\tau + c_0. \tag{3.13}$$

Also define $c_d = |A| - |B|$. Now substitute c_d and Equation 3.13 to Equation 3.11.

$$\begin{aligned}
d_u^2(A,B) &= \int_0^1 \left[(c_2\tau^2 + c_1\tau + c_0)^2 + \theta c_d^2\right] d\tau \\
&= \int_0^1 \left[c_2^2\tau^4 + c_1c_2\tau^3 + c_0c_2\tau^2 + c_1c_2\tau^3 + c_1^2\tau^2 + \right. \\
&\quad \left. c_0c_1\tau + c_0c_2\tau^2 + c_0c_1\tau + c_0^2 + \theta c_d^2\right] d\tau \\
&= \left[1/5c_2^2\tau^5 + 1/4c_1c_2\tau^4 + 1/3c_0c_2\tau^3 + 1/4c_1c_2\tau^4 + 1/3c_1^2\tau^3 + \right. \\
&\quad \left. 1/2c_0c_1\tau^2 + 1/3c_0c_2\tau^3 + 1/2c_0c_1\tau^2 + c_0^2\tau + \theta c_d^2\tau\right]_0^1 \\
&= 1/5c_2^2 + 1/4c_1c_2 + 1/3c_0c_2 + 1/4c_1c_2 + 1/3c_1^2 + \\
&\quad 1/2c_0c_1 + 1/3c_0c_2 + 1/2c_0c_1 + c_0^2 + \theta c_d^2
\end{aligned}$$

This can be reorganized so $d_u^2(A,B)$ can be computed in terms of these coefficients as a quadratic form. The quadratic form can then be decomposed using a Cholesky factorization of the upper-left submatrix:

$$\begin{aligned}
d_u^2(A,B) &= \mathbf{c}^T Q \mathbf{c} \\
&= \begin{bmatrix} c_2 \\ c_1 \\ c_0 \\ c_d \end{bmatrix}^T \begin{bmatrix} 1/5 & 1/4 & 1/3 & 0 \\ 1/4 & 1/3 & 1/2 & 0 \\ 1/3 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & \theta \end{bmatrix} \begin{bmatrix} c_2 \\ c_1 \\ c_0 \\ c_d \end{bmatrix} \\
&= \mathbf{c}^T L L^T \mathbf{c} \\
&= (\mathbf{a} - \mathbf{b})^T L L^T (\mathbf{a} - \mathbf{b}) \\
&= (L^T\mathbf{a} - L^T\mathbf{b})^T \cdot (L^T\mathbf{a} - L^T\mathbf{b}) \\
&= d_E^2(L^T\mathbf{a}, L^T\mathbf{b}).
\end{aligned}$$

where d_E^2 is the square of the ordinary Euclidean distance on 4D points, with the

Cholesky decomposition given exactly by

$$L^T = \begin{bmatrix} \sqrt{5}/5 & \sqrt{5}/4 & \sqrt{5}/3 & 0 \\ 0 & \sqrt{3}/12 & \sqrt{3}/3 & 0 \\ 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & \sqrt{\theta} \end{bmatrix}$$

and with

$$\begin{aligned} \mathbf{a} &= [a_2, a_1, a_0, |A|]^T, \\ \mathbf{b} &= [b_2, b_1, b_0, |B|]^T. \end{aligned}$$

The definition of d_u is similar to the uniform Euclidean distance introduced in [Keo03], but is defined over continuous rather than discrete space, and includes a quadratic penalty term for differences in the lengths of segments.

Now consider further a *minimal polynomial envelope (MPE)* where the constant coefficients of the polynomial approximations are replaced by an interval, for instance $a_0^I = [\underline{a}_0, \overline{a}_0] = [a_0 - h/2, a_0 + h/2]$, where h is called the *radius* of the interval. The intent here is that the MPE bounds the actual data and that we can compute a distance function between features that is a lower (or upper) bound on the actual distance between the data.

Consider the mapping $L^T\mathbf{a}^I = [a_2, a_1, a_0^I, |A|]$. Looking at the form of the matrix L^T, we see that $L^T\mathbf{a}^I$ is a line segment in 4D space oriented along the axis given by the vector $[\sqrt{5}/3, \sqrt{3}/3, 1/3, 0]^T$. Suppose R is a rotation matrix that maps this vector onto one of the axes of the 4D target space, say the x axis. This representation is convenient to use with R-trees and other axis-aligned spatial data structures. Note that $R^T = R^{-1}$, so $R^TR = I$, and also note that rotations preserve the Euclidean distance. We can also set up this rotation so the last coordinate (related to the length

penalty) is unchanged. The mapping is now

$$\begin{aligned} \mathbf{a}_R^I &= [a_x^I, a_y, a_z, a_w]^T && (3.14) \\ &= [[\underline{a}_x, \overline{a}_x], a_y, a_z, a_w]^T && (3.15) \\ &= RL^T\mathbf{a}^I, && (3.16) \end{aligned}$$

where $a_x^I = [\underline{a}_x, \overline{a}_x]$ is an interval and $\mathbf{a}_R^I$ is a line segment aligned with the x axis.

To compute a lower bound on the distance between two quadratic MPEs $\mathbf{a}$ and $\mathbf{b}$, we first map them both into 4D space using $\mathbf{a}_R^I = RL^T\mathbf{a}$ and $\mathbf{b}_R^I = RL^T\mathbf{b}$. We then consider whether these line segments overlap in x, that is if $a_x^I \cap b_x^I \neq \emptyset$. If their x ranges intersect then we compute the distance between the two line segments by computing the $(n-1)$-distance between their other coordinates. If their x ranges do not intersect, then we compute the n-D distance between the closest two endpoints. The result will be a lower bound on the distance between any data curves enclosed by the MPEs. It should be obvious how to extend this analysis to MPEs of any order. In Chapter 4, we will show how to compute upper bounds.

3.2.2 Feature Sequence Representation

Moving from coarse to fine in the scale space, segments may be subdivided into two or more segments because of the appearance of new boundary points. This naturally creates a hierarchical structure as shown in Figure 3.3(b), and this structure can be represented as a tree that is called a *feature tree*.

A feature tree can be a general tree in which each node may have n children where $n \geq 0$. We convert this general tree to a canonical binary tree in which each node can have at most two children (see Algorithm 3.1 and Figure 3.6). We also would like to shorten long unitary branches in which each interval node has only one child and the height of the branch is greater than two. To convert a tree, we use the method ConvertToBinaryTree(*root_node*) given in Algorithm 3.1, where *root_node* is the root of

the tree. An example of a unitary branch is shown in Figure 3.5. The branch a-b-e is shortened to a-e by deleting the node b.

Algorithm 3.1 ConvertToBinaryTree: Convert a general tree to a binary tree without unitary branches.

```
void ConvertToBinaryTree(current_node) {
n ← #children of current_node;
if n > 2 then
  TiltChildren(current_node); // convert to binary branches
end if

child_node ←GetChild(current_node, 1); // GetChild(current_node, i) returns the
i^th child in a post-order traverse

nc ←#children of child_node;
if n = 1 and nc = 1 then
  delete child_node; // remove long unitary branches
end if

for all 1 ≤ i ≤ n and n > 0 do
  ConvertToBinaryTree(GetChild(current_node, i))
end for
}
```

We also represent the nodes of the binary tree in a linear sequence by traversing the binary tree sets in a postfix order. We call such a sequence a *feature sequence*. Each element of the sequence is a data structure called a $Node$ (see Table 3.1). The index of the parent, the left child, and the right child of the current node can be computed by $i - parent$, $i - lchild$, and $i - rchild$, respectively. If $parent = 0$, then the current node is the root. If $lchild = rchild = 0$, then the current node is a leaf. This data structure requires space for 4 integers and $n+2$ floats for a single feature. If we limit each integer to 16 bits and represent each float with 32 bits, a feature requires $32n + 128$ bits.

When storing a feature sequence in the database, we can use the more compact data structure as shown in Table 3.2. Instead of saving offsets to the indices to the parent and the children as shown in Table 3.1, we only need a binary mark to remember if the

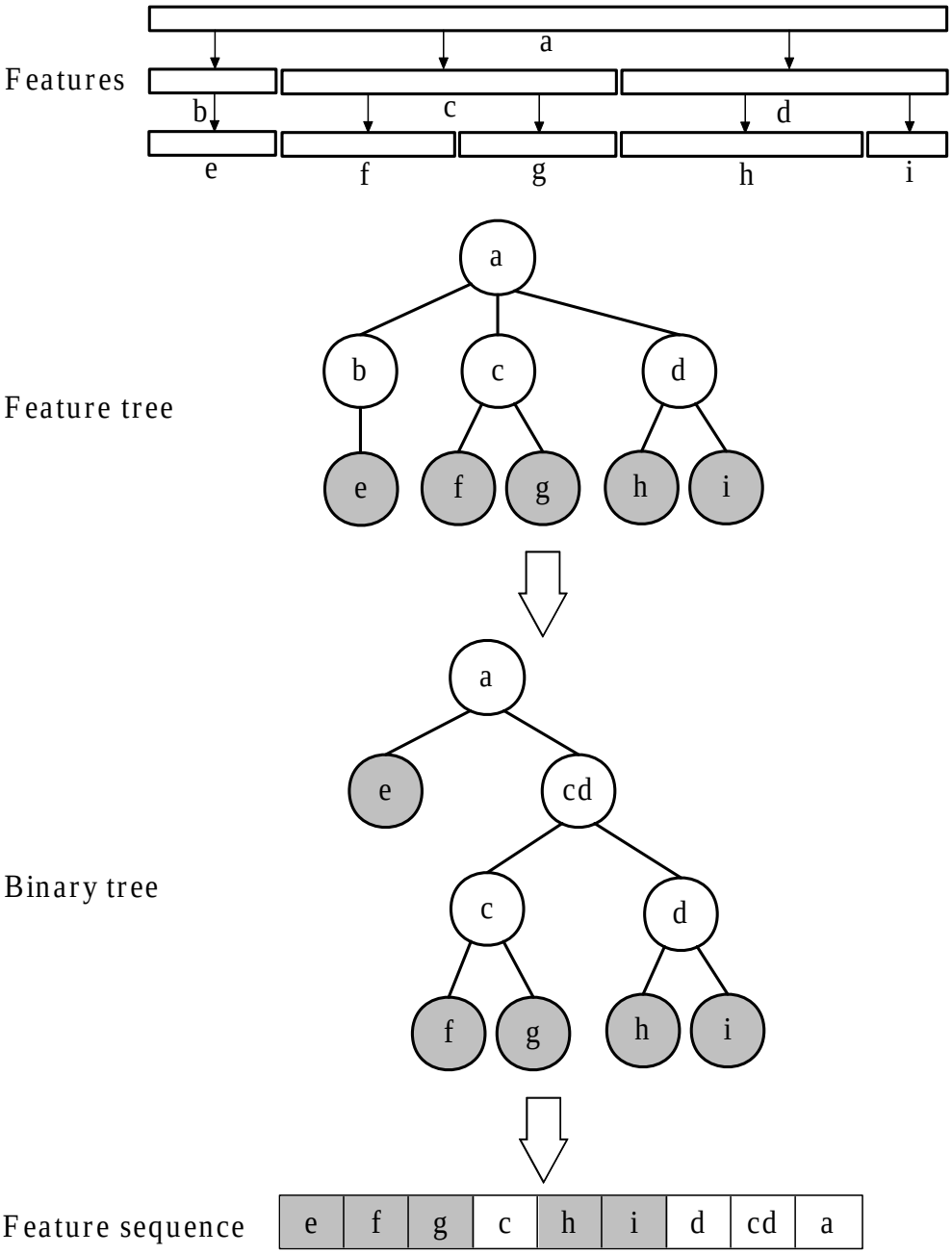


Figure 3.5: The hierarchical features of a time series is represented as a feature tree. The feature tree is converted to a binary tree using Algorithm 3.1. The nodes are represented in a linear sequence by traversing the binary tree in a postfix order. The filled circles are the leaf nodes.

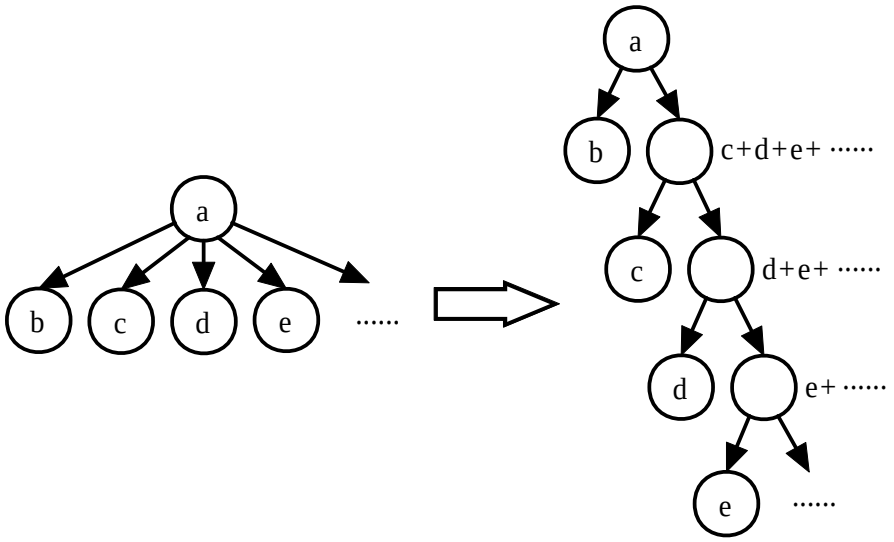


Figure 3.6: Internal nodes are added to convert a two-level subtree with more than two children into a binary subtree. The internal node is the concatenation of both of its leaf nodes.

Table 3.1: The data structure of a node in a feature binary tree. The index of the parent, the left child, and the right child of the current node can be computed by $i - parent$, $i - lchild$, and $i - rhild$, respectively. If $parent = 0$, then the current node is the root. If $lchild = rchild = 0$, then the current node is a leaf.

```
struct Node {
    int parent;       // the offset to the index of the parent
    int lchild;       // the offset to the index of the left child
    int rchild;       // the offset to the index to the right child
    int length;       // the length of the segment
    float a[n + 1];   // the coefficients of n-degree polynomials
    float h;          // the radius of the interval
};
```

Table 3.2: A more compact data structure of a node in a feature binary tree. Instead of saving offsets to the indices of the parent and the children as shown in Table 3.1, only a binary mark is used to remember if the node is a leaf. A stack can be used to reconstruct the binary tree.

```
typedef struct Node {
    boolean is_leaf;   // a mark to determine whether the node is a leaf
    int length;        // the length of the segment in samples
    float a[n + 1];    // the coefficients of n-degree polynomials
    float h;           // the radius of the interval
};
```

node is a leaf. We can then use a stack to reconstruct the binary tree using the fact that the nodes are stored in postfix order and the tree is binary. If the current node is a leaf, we push it into the stack. If the current node is a non-leaf, we pop up two nodes from the stack and associate these two nodes as the children of the current non-leaf node. We push this current non-leaf node into the stack. We scan the sequence from head to tail until the stack is empty. If we again limit each integer to 16 bits and represent each float to 32 bits, using this compact data structure a feature requires $32(n+2)+16+2 = 32n+82$ bits.

3.2.3 Index Construction

To build the index, we select a finest scale (see Figure 3.3) and insert *all* features at every scale of up to this scale into an R-tree index for each time series dataset (see Figure 3.7). Every segment in scale space is represented as an n-D polynomial that can be mapped to an $(n+1)$D line segment in an abstract "feature space"[1]. Euclidean distances in this feature space are then equivalent to distances between polynomials as defined in Equation 3.11. The number of nodes in the R-tree can be increased or decreased by modifying the choice of the finest scale. This flexibility can be used to fit the index into a specific amount of storage space. However, note that a binary tree

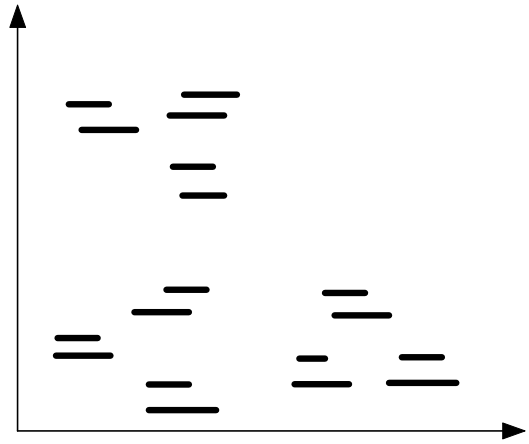
[1]Details can be found in Section 3.2.1

only has $2n-1$ nodes in total if it has n leaves, so inserting all scales only doubles the number of nodes that need to be stored. As we will see, storing parent features in the R-tree enables strong noise immunity.

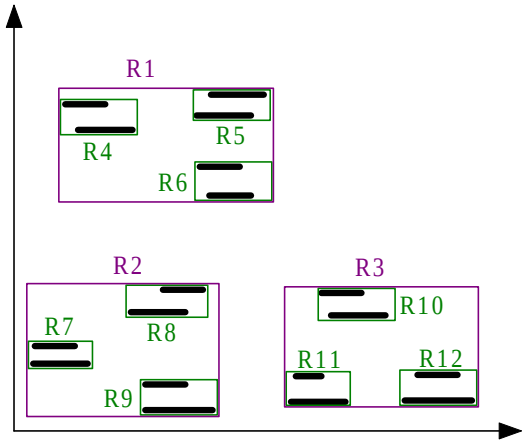
Each time series in a dataset is represented as a feature sequence (encoding the tree in a postfix order). We can associate the R-tree leaves with the features in the feature sequences instead of saving feature representations redundantly. Both the feature sequences of all time series and the R-tree of the dataset are saved as files in secondary storage devices. When the subseries join of two datasets needs to be computed, we load (maybe only the coarser-scale parts of) feature trees of all features in the two datasets and the two R-trees of the two datasets into the main memory (RAM). We then compute the R-tree join and extract parts of the feature sequences as the input to a dynamic programming algorithm (which will be introduced in Section 4.2) to find the subseries join.

3.3 Hierarchical Compression

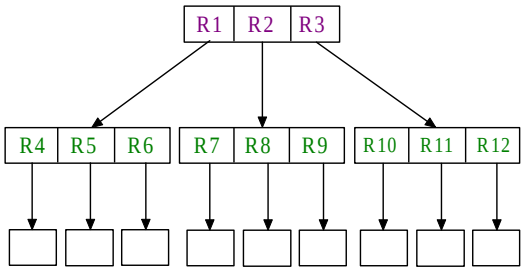
Based on the hierarchical structure generated from the anisotropic diffusion smoothing process, we now present a lossy compression technique for feature sequences, which includes three steps. The first step changes the power basis of the minimal polynomial envelope to a Bézier basis using a linear transformation. The Bézier basis is chosen because all control points of a Bézier spline require the same dynamic range and precision. The power basis requires more precision for the higher-order coefficients than the lower-order coefficients. The second step codes the control points of the Bézier spline using a hierarchical coding method. This method uses the control points in coarser scales to predict the points in finer scales using the de Casteljau subdivision algorithm [Far01]. Only the differences between this prediction and the actual values at the coarser scales are saved. The third step uses arithmetic coding to further encode these differences at one scale. The compression technique presented in this section is a



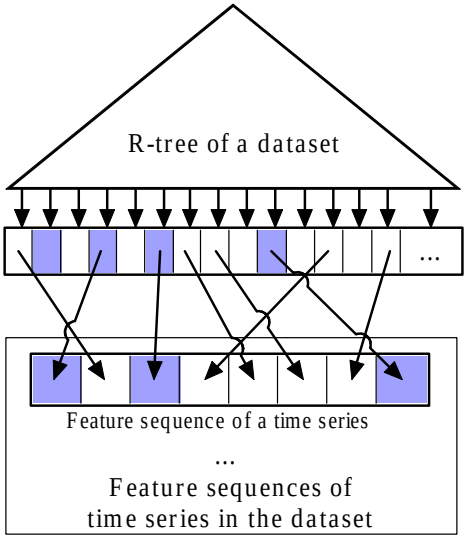
(a) All features at all scales in one dataset. Every segment in scale space is represented as an n-D polynomial that is mapped to an (n+1)D line segments in an abstract "feature space".



(b) Spatial hierarchy of an R-tree of the dataset.



(c) Corresponding nodes in a data structure representing the R-tree.



(d) The R-tree leaves associated with the features in feature sequences.

Figure 3.7: Insert all features at all scales in one dataset into an R-tree. Euclidean distance in feature space are equivalent to distances between polynomials as defined in Equation 3.11. For storage, the R-tree leaves are associated with the features in the feature sequences, instead of saving feature representations redundantly.

generalization of the author's previous work [LM07] that only focused on compression of motion capture data.

A n-degree polynomial

$$P(t) = \sum_{i=0}^{n} a_i t^i \tag{3.17}$$

can be linearly transformed to a Bézier spline

$$\begin{aligned} B(t) &= \sum_{i=0}^{n} b_i B_i^n(t), \\ B_i^n(t) &= \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{n-i} t^i, \end{aligned} \tag{3.18}$$

where $0 \leq t \leq 1$. Consider the specific case of cubic polynomials. Then a cubic polynomial is given relative to the power basis

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0.$$

A cubic Bézier spline is given by

$$\begin{aligned} B(t) &= b_3 t^3 + 3b_2(1-t)t^2 + 3b_1(1-t)^2 t + b_0(1-t)^3 \\ &= (b_3 - 3b_2 + 3b_1 - b_0)t^3 + (3b_2 - 6b_1 + 3b_0)t^2 + (3b_1 - 3b_0)t + b_0. \end{aligned}$$

The transformation from the coefficients of a cubic polynomial to the control points of a cubic Bézier spline is given by

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}. \tag{3.19}$$

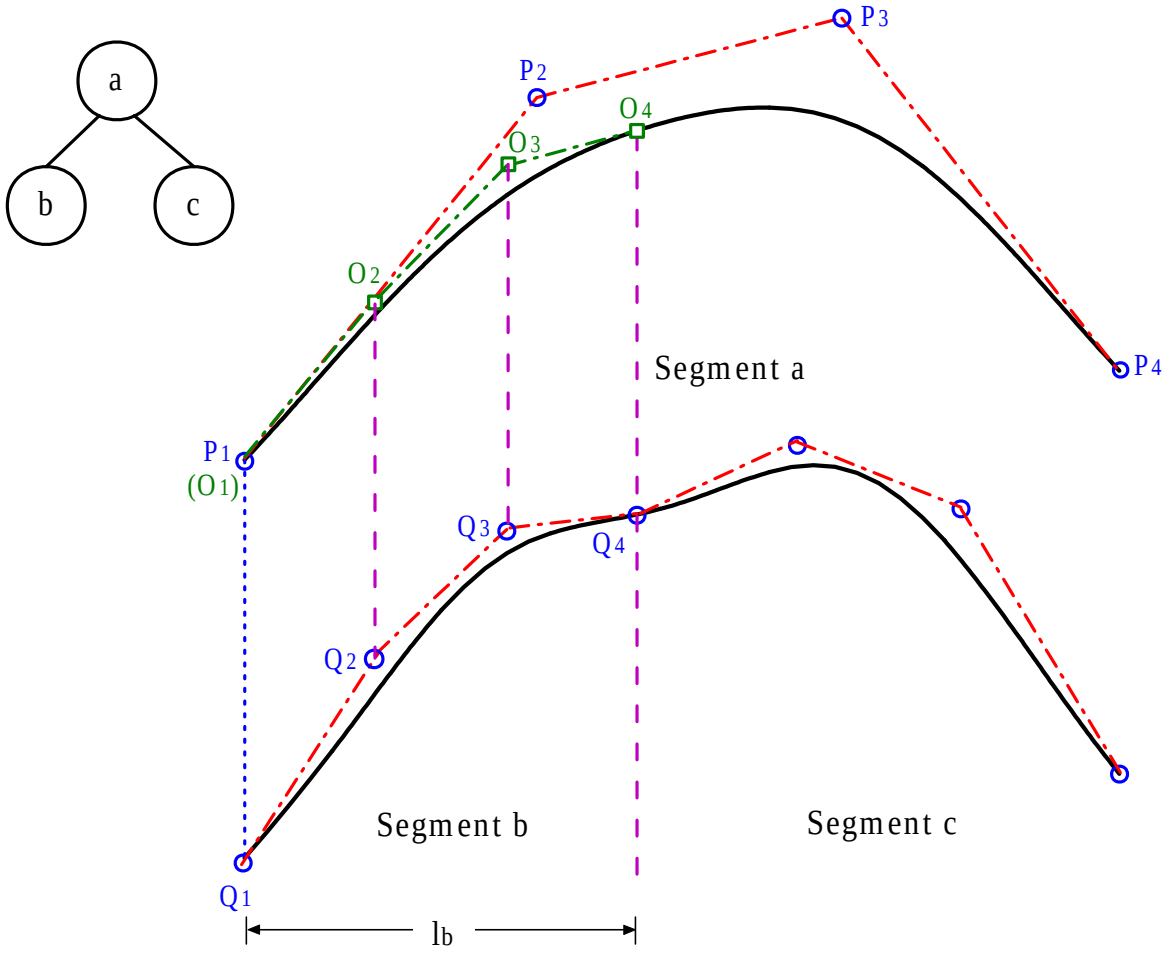


Figure 3.8: Hierarchical differencing of control points of cubic Bézier splines. The segment a is the parent of the segments b and c. We can compute the control points O_1, O_2, O_3, and O_4 of the sub-section of a, that has the same length, ℓ_b, as the length of segment b. Since the points O_i are close to the points Q_i for $1 \leq i \leq 4$, we can use the difference O_i, $\delta_i = Q_i - O_i$, and the offset ℓ_b, to represent Q_i. The points O_i can be computed from the points P_i. Each value δ_i is small enough to be quantized in a small fixed number of bits.

The hierarchical coding of the control points at different levels of the feature tree is illustrated in Figure 3.8. The segment a is the parent of the segments b and c. The control points of the cubic Bézier spline of segment a are P_1, P_2, P_3, and P_4. The control points of the cubic Bézier spline of segment b are Q_1, Q_2, Q_3, and Q_4. We can compute the control points O_1, O_2, O_3, and O_4 of the sub-section of a, that has the same length, ℓ_b, as the length of segment b. Since the points O_i are very close to the points Q_i for $1 \leq i \leq 4$, we can use the difference O_i, $\delta_i = Q_i - O_i$, and the offset ℓ_b, to represent Q_i. The points O_i can be computed by the points P_i. Each value of δ_i is small enough to be quantized in a small fixed number of bits (or fewer, using more sophisticated variable-rate schemes, but this can be done in a post process). We code segment c following the same procedure.

We use an arithmetic coding library to further encode the differences δ_i. Arithmetic coding is a lossless compression technique that gives a variable-length entropy encoding. Compared with other entropy encoding techniques that separate the input into its component symbols and replace each symbol with a code word, such as the Huffman method, arithmetic coding encodes the entire message into a single number, as fraction in $[0, 1)$. By using arithmetic coding, higher compression rates are achieved. The compression rates on actual data will be presented in Chapter 6.

3.4 Summary

In this chapter, a non-uniform segmentation method was proposed based on a scale-space analysis. An approximation of segments identified at each scale was presented using a bounded polynomial approximation. The scale-space analysis is based on anisotropic diffusion and iteratively smoothes each time series, generating a hierarchy of representations from fine to coarse. The smoothed time series at every scale are segmented by the zero-crossings of their second derivatives. The minimal polynomial envelope of each segment and other parameters are used to represent all segments at

all scales in a reduced-dimensionality space, suitable for indexing and compression. A distance function on this scale space was defined that bounds the distance between the fine-scale functions bounded by the polynomial envelopes. A compact data structure was defined to store a feature tree. A compression scheme was also proposed to hierarchically compress feature trees. Indexing and retrieval methods based on this tree structure will be introduced in the following chapter.

Chapter 4

Hierarchical Indexing and Subseries Join

Using the linear transformation derived in the previous chapter, the parameters defining each minimal polynomial envelope can be mapped to axis-aligned line segments in a lower-dimensional space. In particular, in the quadratic case used for the experiments in this thesis, each segment is mapped into 4D space.

In this chapter, indexing and subseries join methods are proposed based on an R-tree join of these axis-aligned segments. Pairs of matching features are found by joining the R-trees of two datasets. Pairs of candidate matching feature sequences can be obtained by counting the number of matching features. Each feature sequence is actually a feature binary tree. A dynamic programming algorithm is developed to calculate the distance between two feature binary trees and to find the alignment between two feature sequences.

4.1 Indexing Using R-trees

The axis-aligned segments representing features can be inserted into an R-tree. To perform a subseries join between two datasets $\mathcal{X}$ and $\mathcal{Y}$, all features are compared in the

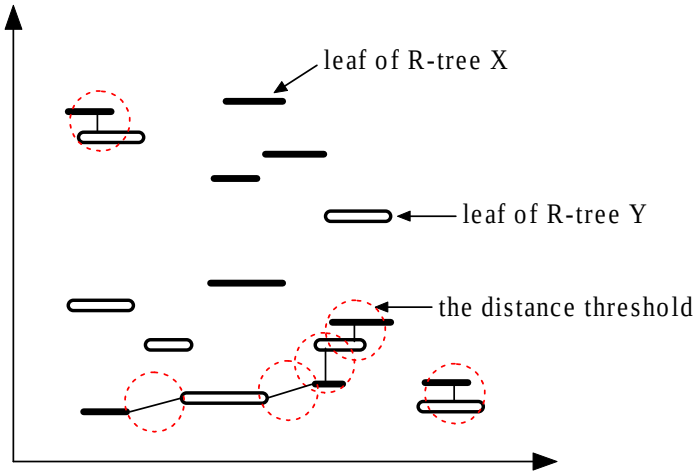


Figure 4.1: Spatial join of the R-trees of two datasets. Note that multiple matches for the same segment are possible. The solid line segments and the hollow line segments are features in two datasets respectively. The polynomial representation of features is mapped to 4D axis-aligned line segments in this abstract space using Equation 3.14.

R-tree index for $\mathcal{X}$ with all features in the R-tree index for $\mathcal{Y}$. The result of such a join is all pairs of features that are closer than some minimum distance from each other. There are many algorithms for performing the spatial join of R-trees [SC03, PD96, CMTV00]. In my implementation, I used the spatial join algorithm proposed in [SC03]. The result of this spatial join of R-trees is the set of all pairs of features from $\mathcal{X}$ and $\mathcal{Y}$ whose minimum distance to each other (this will be defined formally in Equation 4.3) is less than a predefined threshold.

Because the line features are axis-aligned the axis-aligned bounding volumes used in an R-tree data structure can bound them efficiently. The result of the R-tree join process is illustrated in Figure 4.1.

Each feature in the R-tree is represented as a 4D interval aligned with the x-axis:

$$\begin{aligned} \mathbf{a} &= [a_x^I, a_y, a_z, a_w]^T, \\ &= [[\underline{a}_x, \overline{a}_x], a_y, a_z, a_w]^T \end{aligned}$$

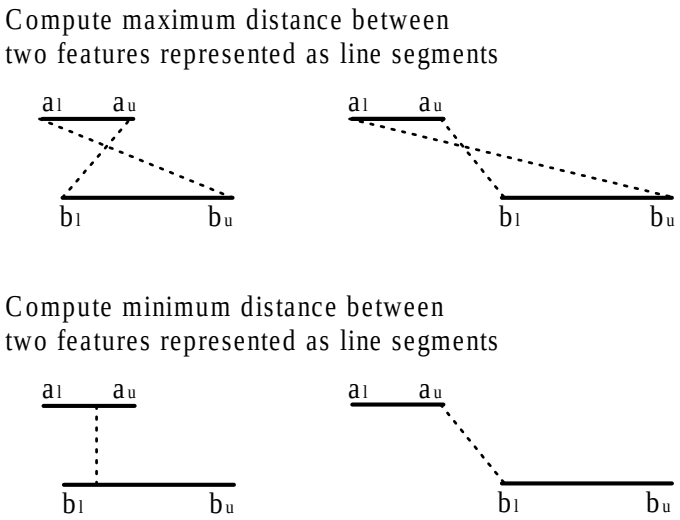


Figure 4.2: Compute maximum and minimum distances between two features represented as line segments.

as defined in Equation 3.14. As shown in Figure 4.2, the maximum distance between two features $\mathbf{a}$ and $\mathbf{b}$ is given by taking the maximum of the distance between end points

$$d_M(\mathbf{a}, \mathbf{b}) = \max\{d_E(\mathbf{a}_\ell, \mathbf{b}_u), d_E(\mathbf{a}_u, \mathbf{b}_\ell)\}, \tag{4.1}$$

where

$$\begin{aligned} \mathbf{a}_\ell &= [\underline{a}_x, a_y, a_z, a_w], \\ \mathbf{a}_u &= [\overline{a}_x, a_y, a_z, a_w], \\ \mathbf{b}_\ell &= [\underline{b}_x, b_y, b_z, b_w], \\ \mathbf{b}_u &= [\overline{b}_x, b_y, b_z, b_w], \end{aligned}$$

and the function d_E is the Euclidean distance, i.e.,

$$d_E(\mathbf{a}_\ell, \mathbf{b}_u) = \sqrt{(\underline{a}_x - \overline{b}_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2 + (a_w - b_w)^2}, \tag{4.2}$$

The minimum distance between two feature points **a** and **b** can be also computed as follows:

$$d_m(\mathbf{a}, \mathbf{b}) = \begin{cases} \sqrt{(a_y - b_y)^2 + (a_z - b_z)^2 + (a_w - b_w)^2} & \text{if } [\underline{a}_x, \overline{a}_x] \cap [\underline{b}_x, \overline{b}_x] \neq \emptyset \\ \min\{d_E(\mathbf{a}_\ell, \mathbf{b}_u), d_E(\mathbf{a}_u, \mathbf{b}_\ell)\} & \text{otherwise.} \end{cases} \tag{4.3}$$

The minimum distance is used in the R-tree join operation as a lower bound to guarantee no false dismissals.

The leaves of R-tree of a dataset are associated with features in the feature sequences of all time series in that dataset, as shown in Figure 4.3. The R-tree join associates, in a pairwise manner, a subset of the leaves of two R-trees. Based on the associated leaves of the R-trees, pairs of feature sequences can be found by counting the number of pairs of matching features from each sequence. If this number is greater than a predefined threshold, these two feature sequences are taken as a pair of candidate matching feature sequences.

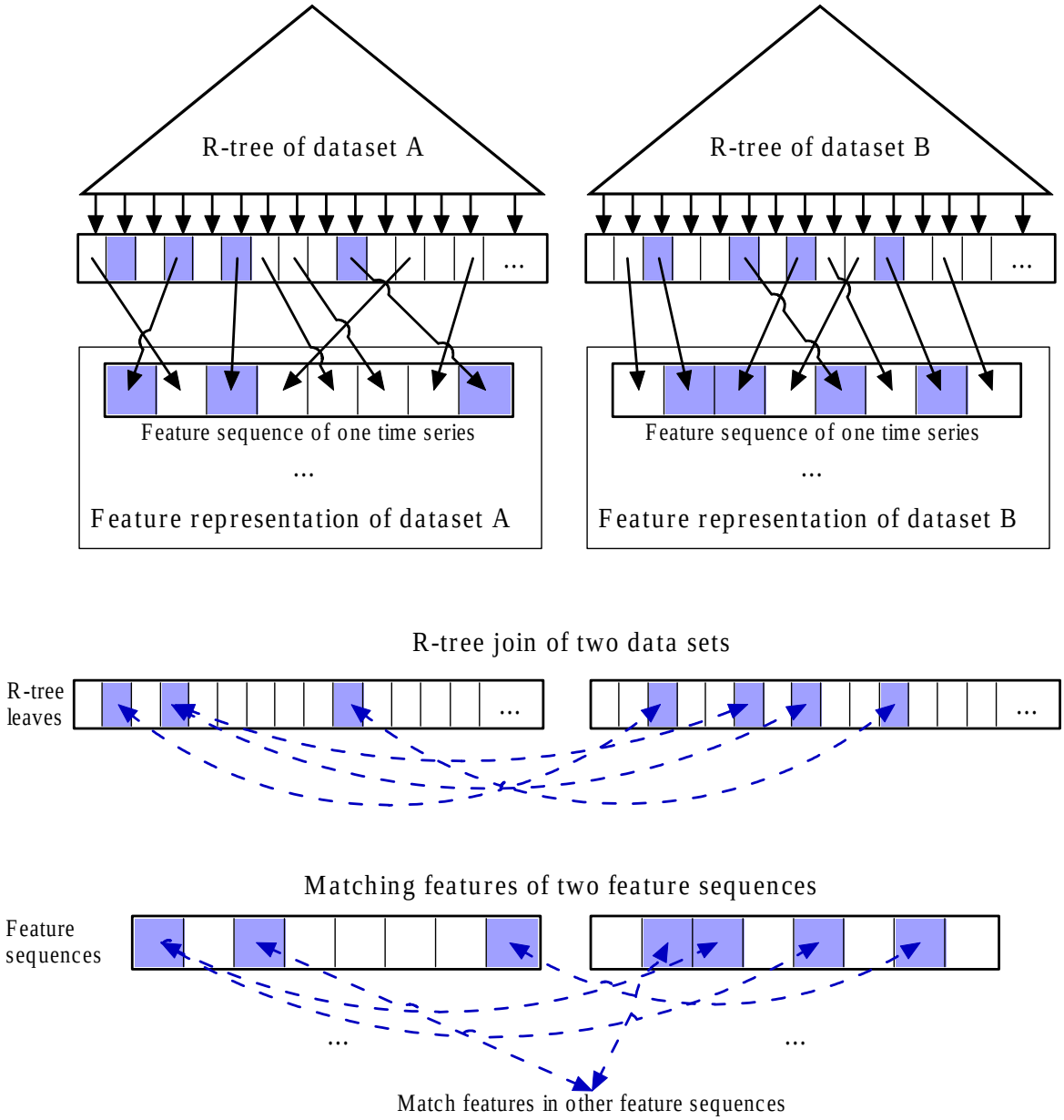


Figure 4.3: The R-tree join of two datasets consists of all pairs of features whose minimum distance is less than a predefined threshold. R-tree leaves are associated with features in the feature sequences of all time series in a dataset. The R-tree join associates some leaves (filled elements) of two R-trees. Based on the associated leaves of R-trees, pairs of matching features (filled elements) can be found in two feature sequences.

4.2 Feature Binary Tree Dynamic Programming

As presented in the previous section, an R-tree join of two R-trees produces a set of pairs of candidate matching feature sequences that contain a number of pairwise feature matches whose minimum distance is below a predefined threshold. However, over a potentially matching subseries, we observe that the minimal distance between one feature and another may be beyond the predefined threshold, possibly due to noise, even if it would otherwise match due to shape. Therefore, the parent of each feature (which has less noise due to smoothing) also needs to be considered. However, such a match of smoothed features is less conclusive than a match at a finer scale.

Since noise will be present, when two feature binary trees are compared and matched to find an alignment we need to use an algorithm that can deal with noise and approximate match appropriately. We assert that the alignment algorithm should have the following properties:

- It should prefer matching at finer resolutions when possible.
- It should tolerate at least two kinds of noise:
  - *Impulsive noise*. Tolerating impulsive noise (or outliners) means allowing small gaps in the match.
  - *Additive noise*. Tolerating additive noise means allowing matching of smoothed data, i.e., "parent" features, but only when matching at a finer scale fails.
- It should allow subsequence match and should not have to process all data in a sequence to do so.
- It should find the alignment with the smallest worst-case distance (min-max), if alternative alignments are possible.

I have developed an algorithm that satisfies these goals. This algorithm uses dynamic programming to optimally select among multiple alternatives for alignment,

but uses heuristics to minimize the number of choices that have to be considered, in particular to focus the alignment on finer scales when possible. It works as follows:

1. The alignment scans from left to right, starting from the left-most matching leaves of both feature sequences.

2. If both children match, the alignment ignores their parents.

3. If neither child matches, the alignment checks its parent.

4. If one child matches and the other child does not match,

    - if the unmatched child is small enough to be considered a "gap", the alignment goes through this gap and can ignore its parent.
    - if the unmatched child is not small enough to be a gap, the alignment ignores both children and checks their parent.
    - Dynamic programming is used to select among other alternatives.

Rules 2-4 constitute a heuristic that favors fine-scale matches over coarse-scale matches. To illustrate this algorithm, an example is shown in Figure 4.4. In feature binary trees X and Y, the pairs of matching features returned by the R-tree join operation are {b, b'}, {d, d'}, {e, e'}, {f, f'}, {f, h'}, {g, i'}, {h, j'}, and {i, k'}. First, a minimal subtree is extracted that contains the features from the left-most matching feature b to the right-most matching feature g and all their parents. This procedure generates a feature subsequence X'. The feature subsequence Y' is generated for the feature sequence Y using the same procedure.

The alignment starts from the pair of matching features b and b'. The features c and c' do not match, but there are options that the alignment algorithm can consider. If c and c' are features with small lengths (small segments), which can be considered gaps (small segments of impulsive noise), the alignment goes through {b, b'} and {c, c'} and can ignore their parents d and d'. If either c or c' is not small enough to be

a gap, the alignment will check the parents d and d', and can ignore {b, b'} and {c, c'}. Another case shown in this figure illustrates what happens when both pairs of children match, for example, f matches h' and g matches i'. The alignment does not need to consider the parents h and j', even though the R-tree says they match. These heuristics eliminate many choices for the alignment. However, note that many-to-one (and many-to-many) matches are possible, such as {f, f'} and {f, h'}, which can allow alternative alignments. Therefore, the alignment algorithm is designed as a dynamic programming algorithm that can choose among the various alignment possibilities when such multiple matches are present.

Dynamic programming using a warping score matrix is a classic way to solve sequence alignment problems and was introduced in Section 2.2.4. However, we are matching trees rather than flat sequences so we cannot use this algorithm directly. Based on warping matrix dynamic programming, I have developed a *Feature Binary Tree Dynamic Programming (FBTDP)* algorithm. The description of FBTDP is as follows: Given two feature sequences, $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_n)$, FBTDP constructs a warping distance matrix WM using a recurrence. Each option in the recurrence represents one possible local choice for alignment. The recurrence propagates the best alignment through the matrix by choosing the best local extension to the alignment. The recurrence is

$$WM(i,j) = \max \begin{cases} 0 & \textit{// no match} \\ WM(i - p(a_i), j) + \alpha(a_i) & \textit{// } a_i \textit{ is a gap} \\ WM(i, j - p(b_j)) + \alpha(b_j) & \textit{// } b_j \textit{ is a gap} \\ WM(i - p(a_i), j - p(b_j)) + \phi(a_i, b_j) & \textit{// } a_i \textit{ and } b_j \textit{ match.} \end{cases} \tag{4.4}$$

where the boundary conditions of this recurrence are $WM(i,0) = WM(0,j) = 0$ and

$$p(a_i) = \begin{cases} n_c(a_i) - 2 & \text{if } a_i \text{ is the left child} \\ n_c(a_i) - 1 & \text{if } a_i \text{ is the right child,} \end{cases} \tag{4.5}$$

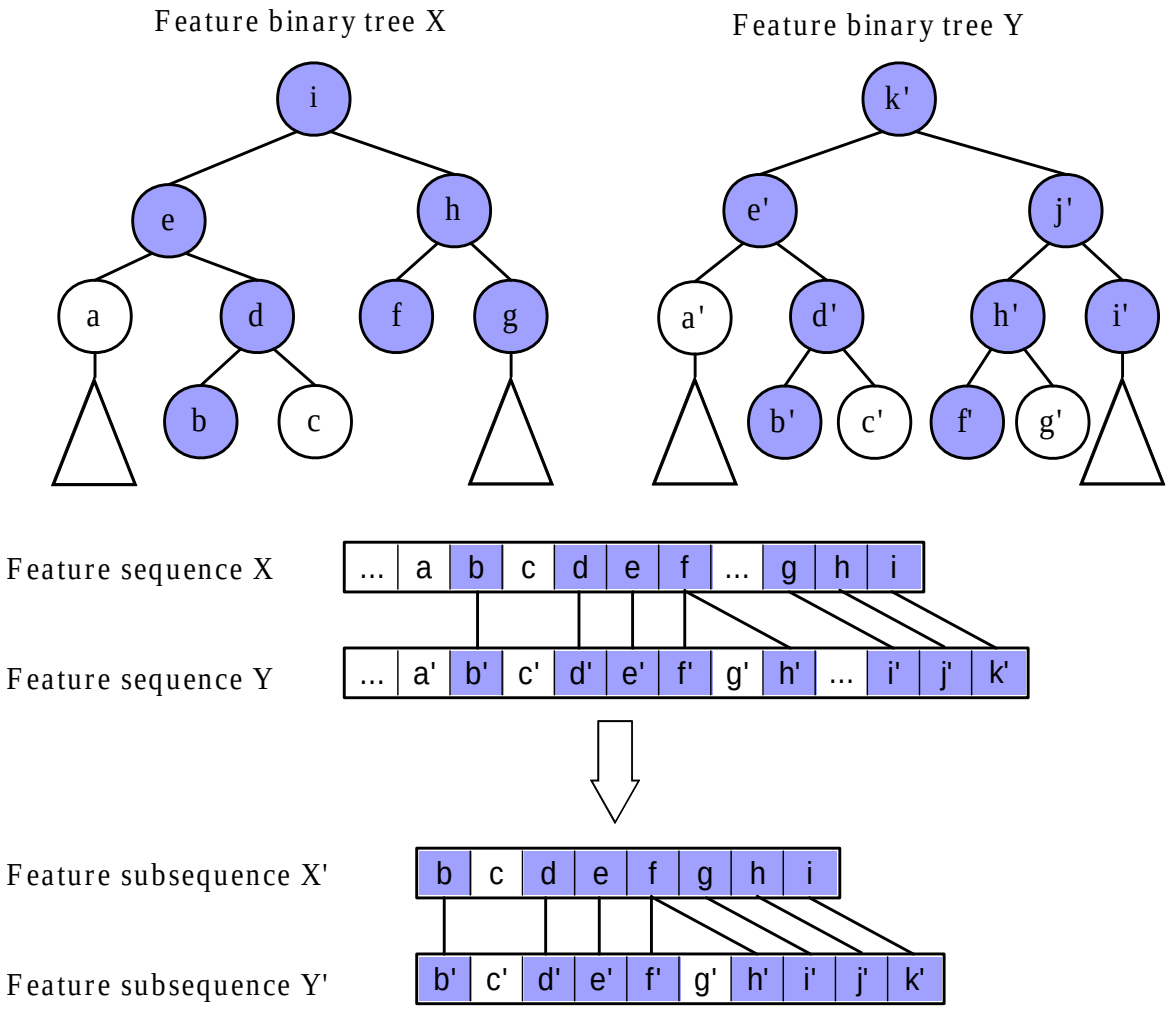


Figure 4.4: Two feature binary trees and pairs of matching features (filled elements) found by an R-tree join operation. A subtree is extracted that contains the nodes from the left-most matching feature b to the right-most matching feature g and all their parents. This procedure generates a feature subsequence X'. The feature subsequence Y' is generated for the feature sequence Y using the same procedure. Note that multiple-to-one matches are possible, such as {f, f'} and {f, h'}.

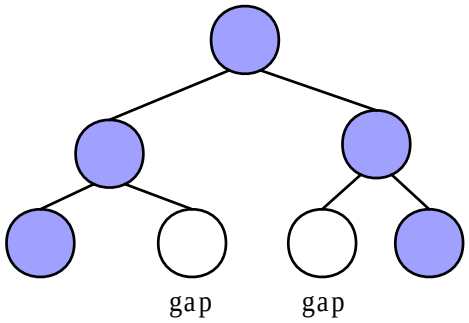


Figure 4.5: The unfilled elements are gaps. The number of consecutive gaps at the same level in a tree is at most two. In other words, at most two gaps can be adjacent. Therefore, the bound on the maximum gap size is $2G$.

likewise for $p(b_j)$ and b_j. The function $p(a_i)$ returns the offset to the sibling or "cousin" node at the same level in the tree. The function $n_c(a_i)$ returns the total number of children of a_i. The function ϕ is a heuristic scoring function that we will discuss later. The gap penalty function α is defined as follows:

$$\alpha(g) = \begin{cases} 1/2 & \text{if } |g| \leq G \\ -\infty & \text{otherwise,} \end{cases} \tag{4.6}$$

where $2G$ is the maximum length of a gap that can be tolerated. A larger value of G increases the tolerance of the algorithm to impulsive noise. A gap is only allowed for features that do not match any other feature. Note that gaps are only allowed if a sibling matches. The bound on the maximum gap size is thus $2G$ since the situation in Figure 4.5 is possible.

Every element $WM(i, j)$ is an accumulative warping score that reflects the alignment cost up to position (i, j). The element $WM(m, n)$ is the total accumulative warping score of the alignment of two feature subsequences. Both the space complexity and the computational complexity of FBTDP are $O(mn)$.

We will now discuss the heuristic scoring function ϕ. Since each feature sequence is a binary tree, if an internal node has only one child, then this internal node must be

the root. This occurs only in a feature binary tree with two nodes. Note that this is rare since it can only occur when the original time series data is constant. We therefore treat this as a special case and eliminate it from further discussion. For any feature binary tree with more than two nodes, each internal node has two children. Therefore, to define the scoring function $\phi(i, j)$, the following cases need to be considered. Note that the scoring function is only ever called on matching features.

1. If both a_i and b_j are leaves, then these two nodes are compared directly using the following function:

$$\phi(a_i, b_j) = \text{score}(a_i, b_j), \tag{4.7}$$

$$\text{score}(a_i, b_j) = \begin{cases} 1 & \text{if } d_m(a_i, b_j) \leq \epsilon_f \\ -\infty & \text{otherwise.} \end{cases} \tag{4.8}$$

   Note that this just accepts the matches already identified by the R-tree. Also, the score is independent of the length of the feature or the distance between them, for reasons we will discuss later.

2. If a_i is an internal node and b_j is a leaf, the children of a_i need to be considered by the alignment algorithm too, in case one is a gap:

$$\phi(a_i, b_j) = \max \begin{cases} \text{score}(a_i, b_j) \\ \text{score}(a_{i-a_i.lchild}, b_j) + \alpha(a_{i-a_i.rchild}) \\ \text{score}(a_{i-a_i.rchild}, b_j) + \alpha(a_{i-a_i.lchild}). \end{cases} \tag{4.9}$$

3. Likewise, if a_i is a leaf and b_j is an internal node, the children of b_j need to be considered:

$$\phi(a_i, b_j) = \max \begin{cases} \text{score}(a_i, b_j) \\ \text{score}(a_i, b_{j-b_j.lchild}) + \alpha(b_{j-b_j.rchild}) \\ \text{score}(a_i, b_{j-b_j.rchild}) + \alpha(b_{j-b_j.lchild}). \end{cases} \tag{4.10}$$

4. If both a_i and b_j are internal nodes, the children of both a_i and b_j need to be considered by the alignment algorithm:

$$\phi(a_i, b_j) = \max \begin{cases} \text{score}(a_i, b_j) \\ \text{score}(a_{i-a_i.lchild}, b_{j-b_j.lchild}) + \text{score}(a_{i-a_i.rchild}, b_{j-b_j.rchild}). \end{cases} \tag{4.11}$$

The indices *lchild* and *rchild* were defined in Table 3.1 and are stored in the data structure representing feature.

Each element $WM(i, j)$ in WM is an accumulative warping score that reflects the alignment cost up to this position (i, j). The element $WM(m, n)$ is the total accumulative warping score of the alignment of two feature sequences. As noted, the per-feature score is intentionally independent of length or minimum distance so that the warping score will be higher for matching many children rather than one parent. This algorithm causes the algorithm to prefer an alignment at a finer scale if one can be found. Likewise, a smoothed parent tends to have a lower minimum distance than a more precise fine-scale feature, so we do not include this in our score to avoid a bias towards smoothed features.

If the value of $WM(m, n)$ is above a predefined threshold, a trace-back step is taken to find the best alignment based on the heuristic scoring function. For each pair of elements $\{a_i, b_j\}$ in the alignment we compute the accumulative maximal distance

$$d_{AM}(A, B) = \sum \{d_M(a_i, b_j) | \{a_i, b_j\} \text{ in the alignment}\}, \tag{4.12}$$

using the maximal distance bound d_M that was defined in Equation 4.1. The distance function d_{AM} computes the min-max accumulative distance heuristically, and it is this distance that I used in my experimental results to compare the similarity of two feature sequences A and B. The accuracy of the FBTDP algorithm will be shown empirically in the following chapters.

FBTDP is not guaranteed to return the optimal answer. The results computed by

thresholding d_{AM} do not contain false positives, but may contain false dismissals. We can compute the accumulative minimal distance

$$d_{Am}(A, B) = \sum \{d_m(a_i, b_j) | \{a_i, b_j\} \text{ in the alignment}\}, \tag{4.13}$$

using the minimal distance bound d_m that was defined in Equation 4.3. The results computed by thresholding d_{Am} do not contain false dismissals, but may contain false positives. The optimal answer should be some value in between the results returned by d_AM and d_Am. Therefore, although FBTDP does not compute the optimum it does bound it.

The computational complexity of FBTDP is quadratic. However, in my prototype system implementation, I constrained the alignment along the levels of the tree to be at most three. In other words, if three levels of nodes do not match, then FBTDP returns the current alignment as a subseries join result, and starts to find a new alignment in the remaining subtrees. This constraint improves the computational complexity to be linear.

4.3 Summary

This chapter presented an indexing and alignment scheme for binary feature trees. The features at all scales are indexed in an R-tree. Pairs of candidate matching feature sequences are obtained by the matching features returned from spatially joining the R-trees of two datasets. A dynamic programming algorithm was proposed to compute the min-max and max-min matching distance between binary feature trees.

Chapter 5

Experimental Evaluations

To validate the proposed approach in the previous chapters, accuracy and performance relative to other approaches needs to be determined. However, accuracy is based on the definition of similarity. As discussed before, the definition of similarity is ambiguous and subjective, varying with different data formats, tasks, and domains. Also, the bound on this metric using our proposed algorithm is not exact, so we have to evaluate the impact of this approximation. Several recent approaches for subseries matching have been selected as baseline approaches. The proposed approach is compared against them to evaluate effectiveness. To guarantee that the baseline approaches produce correct results, synthesized data are generated from a real dataset [Keo06a] in such a way that the "correct" matches are known in advance. However, it should be emphasized that none of the selected baselines is a gold standard. Also, only the accuracy of "match" can be compared, since the proposed definition of "join" is new.

5.1 Experimental Setup

All time series data are extracted or synthesized from the UCR Time Series Data Mining Archive [Keo06a]. This archive contains different datasets, such as stock prices, audio, and trajectories. The experiments were run using a PC with Linux Kernel 2.6,

Table 5.1: Time series selected from the UCR Time Series Data Mining Archive [Keo06a] as testing datasets.

NO.	**Name**	**Length**	**NO.**	**Name**	**Length**
1	attas	1023	21	network	180000
2	ballbeam	1000	22	ocean shear	4095
3	balloon	2001	23	packet	360000
4	buoy sensor	13990	24	pHdata	2001
5	burst	9382	25	power data	35040
6	burstin	50000	26	powerplant	2400
7	chaotic	1800	27	random walk	65536
8	cstr	7500	28	realitycheck	1000
9	darwin	1400	29	robot arm	1024
10	earthquake	4095	30	shuttle	1000
11	EEG heart rate	7200	31	soiltemp	2305
12	evaporator	6305	32	speech	1019
13	fluid dynamics	10000	33	spot extrates	2567
14	flutter	1024	34	steamgen	9600
15	foetal ecg	2500	35	sunspot	2899
16	glassfurnace	1247	36	tickwise	279113
17	infrasound beamed	8191	37	tide	8745
18	koski ecg	144002	38	water	2192
19	leleccum	4320	39	wind	6574
20	memory	6874	40	winding	2500

Table 5.2: The parameters used in the experiments.

Parameter	Value	Description
ϵ_b	0.01	Threshold used in the segmentation (see Equation 3.10)
θ	0.1	Weight used in the distance metric (see Equation 3.11)
G	5	Threshold for gaps (see Equation 4.6)
ϵ_f	2	Threshold for minimum distances (see Equation 4.7)

1GB of RAM, and a Pentium IV 3.0GHz CPU. The prototype systems were implemented in C++ and the RapidMind Development Platform [Rap]. To fit a constrained amount of memory, the finest scale at $\tau = 500$ was selected for indexing. Although this database is relatively small, it has the advantage of being a standard test case for which results from other algorithms are available in the literature, and is publically available, allowing future work to build comparisons with our results.

A set of 40 time series (see Table 5.1) were extracted from the UCR Time Series Data Mining Archive. This dataset will be called $\mathbb{D}$. The total number of samples in $\mathbb{D}$ is $1,091,465$. The average number of samples per time series in $\mathbb{D}$ is $27,287$. The dataset $\mathbb{D}$ is used to generate other synthetic testing datasets. The synthetic data is used to guarantee that the baseline approaches produce correct results. Table 5.2 lists the parameters used in the experiments.

5.2 Accuracy Evaluation for Subseries Matching

For accuracy evaluation, the proposed approach (called NSDP[1] in the rest of the thesis) is compared against two baseline approaches, shift-and-compare scanning using the Euclidean distance (called SSE) and shift-and-compare scanning using DTW (called SSD).

[1]The name of the proposed approach NSDP is formed from the key words *Non-uniform Segmentation* and *Dynamic Programming*

From each original time series in $\mathbb{D}$, 50 subseries are randomly extracted with their length ranging from 100 to 1000 samples. For each element x in each subseries X', i.e., $x \in X'$, uniform random noise was added to generate a query time series Q with each element $q \in Q$ and

$$q = x + \mu(\max(X') - \min(X')), \tag{5.1}$$

where μ is a uniform random value with $-0.3 \leq \mu \leq 0.3$. This generates 40 query datasets each of which contains 50 short query time series. These datasets, called $Q^e = (Q_1^e, Q_2^e, \ldots, Q_{40}^e)$, are used in experiments to compare NSDP against SSE.

Each time series Q_i^e belonging to the query dataset Q^e is also uniformly scaled using nearest-neighbor interpolation to get 40 new query datasets $Q^d = (Q_1^d, Q_2^d, \ldots, Q_{40}^d)$, i.e., for each element $q_j \in Q_i^e$,

$$q'_j = q_{\lceil j/(\gamma) \rceil}, \tag{5.2}$$

where $q'_j \in Q_i^d \in Q^d$ and γ is a random value with $1 \leq |\gamma| \leq 1.2$. We used nearest-neighbor interpolation (rather than linear interpolation) to be consistent with previous work. Note that since nearest-neighbor interpolation duplicates samples upon expansion, it is consistent with the alignment generated by SSD. This actually favors SSD over NSDP. These datasets Q^d are used for comparing NSDP against SSD.

Given an element $q \in Q_o$ or a scaled element $q' \in Q_s$, let $S_b \subseteq X \in \mathbb{D}$ be the best matching time series returned by the baseline approach (SSE or SSD), and let $S_o \subseteq X \in \mathbb{D}$ be the best matching time series returned by NSDP. Two metrics are used to measure the empirical accuracy:

$$e = \begin{cases} 1 & \text{if } S_b \cap S_o = \emptyset \\ 0 & \text{otherwise,} \end{cases} \tag{5.3}$$

and

$$p = \frac{|S_b \setminus S_o|}{|S_b|}. \tag{5.4}$$

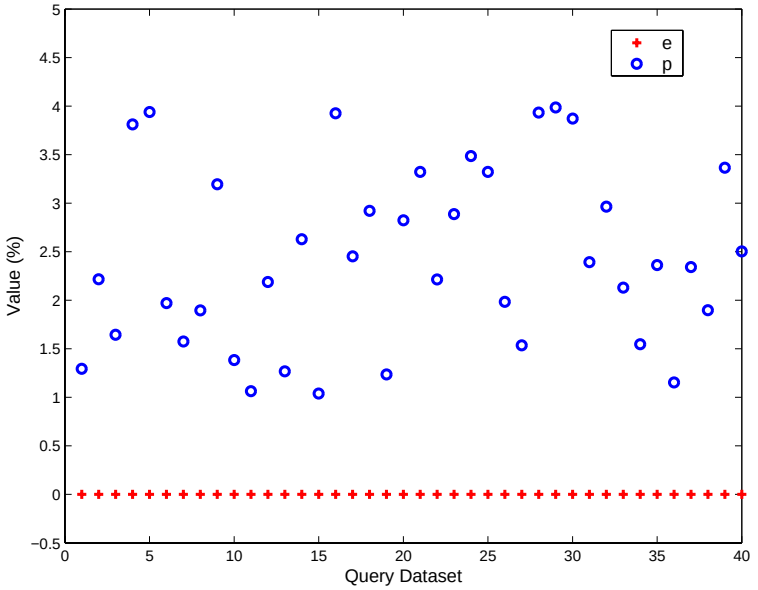
The metric e returns 1 if the subseries found by NSDP does not overlap that found by the baseline; otherwise, it returns 0. In other words, e counts disagreements between two approaches. The metric p computes the percentage of number of items in each subseries that are found by the baseline but not found by NSDP.

The experimental results for all 40 datasets are shown in Figure 5.1. The values of p and e are average values over 50 queries for each time series in $\mathbb{D}$ using the baseline approaches SSE and SSD respectively. In the experiments, all the values of e are 0. This empirically demonstrates that NSDP produces no false dismissals when comparing against the baselines, at least for this data. This is reassuring, but since the similarity measure is different, in general we cannot guarantee this result.

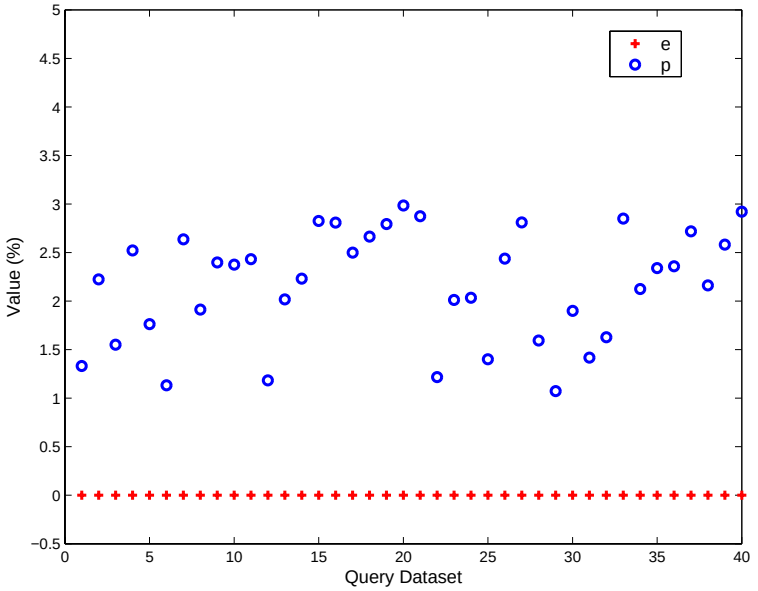
The high percentages of the overlaps of the results found by NSDP and the baselines also demonstrates that NSDP finds similar matching subseries when comparing against the baselines. Some subseries matching results returned by SSE, SSD, and NSDP are shown in Figure 5.2.

In the above description, the results returned by SSE and SSD are assumed to be the "correct" matches. However, this assumption is not always the case. Counter-examples are shown in Figure 5.3. Perceptually, the time series Q is more similar to the time series X than the time series Y, because X is a uniformly scaled version of Q. However, both SSE and SSD approaches match Q against Y. These examples show that SSE and SSD are sensitive to large length differences and impulsive noise. In contrast, NSDP can tolerate both large length differences and both additive and impulsive noise. To better evaluate the proposed approach, therefore, we need a better baseline.

In next section, a more robust approach, an approximation of the *information distance*, will be introduced as a baseline approach to evaluate the effectiveness of the proposed approach in the context of classification.

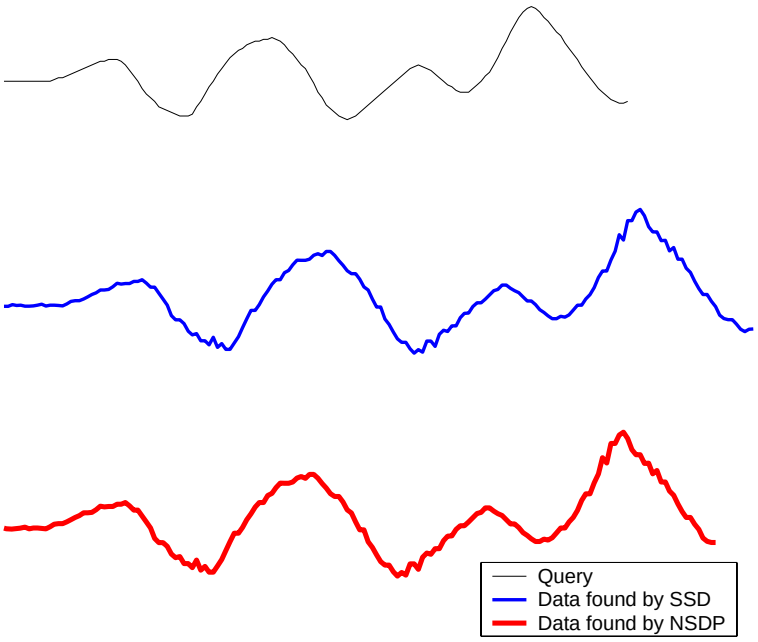


(a) NSDP compares with SSE.

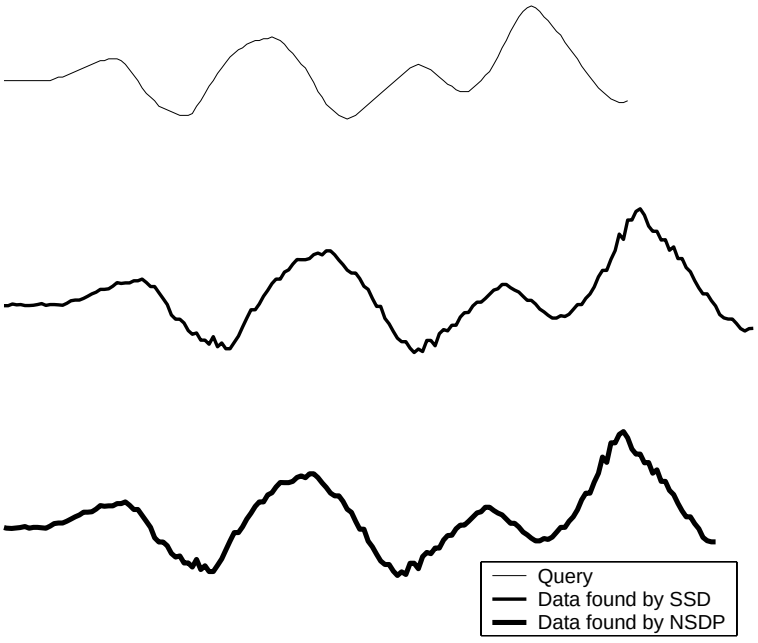


(b) NSDP compares with SSD.

Figure 5.1: Accuracy evaluation for subseries matching. The metric e returns 1 if the subseries is found by the proposed approach but is not found by the baseline; otherwise, it returns 0. The metric p computes the percentage of number of items in each subseries that is found by the proposed approach but is not found by the baseline. The values of p and e are the average values over 50 queries for each time series in $\mathbb{D}$ using the baseline approaches SSE and SSD respectively. In the experiments, all the values of e are 0.

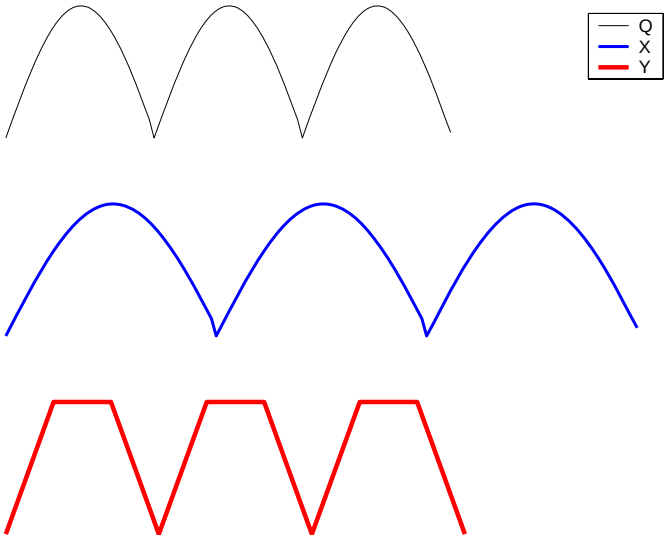


(a) Subseries matching results returned by SSE and NSDP.

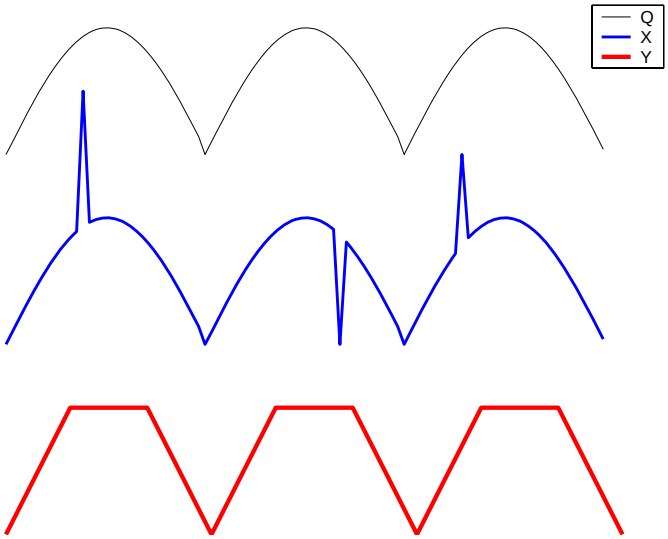


(b) Subseries matching results returned by SSE and NSDP.

Figure 5.2: Subseries matching results returned by SSE, SSD, and NSDP.



(a) Given three time series Q, X, and Y, the distances between two time series returned by SSE, SSD and NSDP satisfy: $SSE(Q,X) > SSE(Q,Y)$, $SSD(Q,X) > SSD(Q,Y)$, $NSDP(Q,X) < NSDP(Q,Y)$.



(b) Given three time series Q', X', and Y', the distances between two time series returned by SSE, SSD and NSDP satisfy: $SSE(Q',X') > SSE(Q',Y')$, $SSD(Q',X') > SSD(Q',Y')$, $NSDP(Q',X') < NSDP(Q',Y')$.

Figure 5.3: The functions $SSE(X,Y)$ and $SSD(X,Y)$ compute the distances using SSE and SSD respectively. The results computed by SSE and SSD are opposite to intuition.

5.3 Accuracy Evaluation for Classification

In this section, background on *information distance* will be introduced first. The experimental results of classification of time series using a baseline approach based on information distance and compared with the NSDP approach are then reported.

5.3.1 Information Distance Metric

Similarity measures determine if one dataset is like another dataset. A dataset commonly belongs to a certain type: music data, financial data, genomic data, etc. Feature-based measures exploit special features of the data related to the specific domain. For example, the features of music data can be extracted, related to pitch, rhythm, harmony etc. However, feature-based measures require domain-related knowledge and are sensitive to the accuracy of feature abstraction. Some non-domain specific measures exist, such as Hamming distance, Euclidean distance, edit distance, and alignment distance. These are non-feature measures and work well in many different domains, but they only account for the differences between datasets, not for their commonalities.

Information distance is a universal similarity distance that does not use subject-specific features or require domain-specific background knowledge. The *normalized information distance* that is based on the *Kolmogorov complexity* has been proven to be optimal [LV97]. The idea behind Kolmogorov complexity is that if two datasets are more similar, then one can be more succinctly described if we are given the other. The Kolmogorov complexity is a measure of absolute information distance. A brief introduction to the Kolmogorov complexity will be given in this section. For more details, the reader can refer to the paper [Li07] by Li and the paper [ZHZL07] by Zhang et al.

The "energy" to convert between two datasets x and y is defined as the shortest

program converting x to y and vice versa. The cost of conversion between x and y is

$$E(x, y) = \min\{|p| : U(x, p) = y, U(y, p) = x\}, \tag{5.5}$$

where p is any binary program executed on U, a universal Turing machine. The Kolmogorov complexity of x conditioned on y is the length of the shortest program that outputs x with input y,

$$K(x|y) = \min_{p}\{|p| : U(p, y) = x\}. \tag{5.6}$$

The Kolmogorov complexity of x is the length of the shortest program that outputs x,

$$K(x) = \min_{p}\{|p| : U(p) = x\}. \tag{5.7}$$

The max information distance [BGL$^+$98] between two objects x and y is

$$D_{\max}(x, y) = E(x, y) = \max\{K(x|y), K(y|x)\}. \tag{5.8}$$

This distance function is a metric since it satisfies positivity, symmetry, and the triangle inequality [BGL$^+$98]. The max distance is also an optimal distance. The *normalized information distance* was introduced by Li *et. al* [LCL$^+$04]:

$$d_{\max}(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \tag{5.9}$$

It has been proven that the normalized information distance is also a metric that satisfies the triangle inequality [LCL$^+$04]. The normalized information distance has the following properties:

1. It is a non-trivial metric. That is, it is nonnegative, symmetric, satisfies the triangle inequality, and is zero if and only if the objects are identical.

2. It is universal. That is, if x and y are close in any computable sense then they are close under $d_{\max}(x, y)$.

3. Unfortunately, it is also not computable, because of the non-computability of $K(x)$.

Also, the normalized information distance involves the impact of irrelevant information [Li07]. That is to say, two time series that have the same relevant part to a query may have different $d_{\max}$ values because of differences in irrelevant parts. Therefore, the normalized minimum distance was proposed by Li [Li07]:

$$d_{\min}(x, y) = \frac{\min\{K(x|y), K(y|x)\}}{\min\{K(x), K(y)\}}. \tag{5.10}$$

It has been proven that the distance $d_{\min}$ is also universal and $d_{\min} \leq d_{\max}$. Unfortunately, like $d_{\max}$, the distance $d_{\min}$ is also not computable. Unlike $d_{\max}$, the distance $d_{\min}$ computes the similarity of local matching. However, since $d_{\min}$ does not satisfy the triangle inequality, the distance $d_{\min}$ is not a metric.

The fact that $d_{\max}$ and $d_{\min}$ are non-computable is frustrating, but fortunately there exists reasonable approximations. Cilibrasi and Vitányi [CV05] presented an *approximate normalized information distance (ANID)* that approximates the optimality of the Kolmogorov complexity. ANID is based on the fact that two objects are deemed close if one can be significantly "compressed" given the information in the other. In particular, the compressed size of the data $C(x)$ can used as an approximation to $K(x)$ [CV05]. Common compressors used to evaluate this metric are bzip2 and gzip. Replacing $K(x)$ with $C(x)$, Equation 5.9 turns into the normalized compression distance:

$$\text{ANID}(x, y) = \frac{\max\{C(x|y), C(y|x)\}}{\max\{C(x), C(y)\}}, \tag{5.11}$$

where $C(x|y)$ compresses x given y.

5.3.2 Classification Results

From the dataset $\mathbb{D}$ in Table 5.1, three groups of datasets are generated by uniformly scaling, adding additive noise to, and adding impulsive noise to the original time series. These three groups of datasets are used to test the robustness of NSDP, ANID, and DTW to different uniform length scaling, additive noise, and impulsive noise.

Robustness to Uniform Scale

For each time series $X \in \mathbb{D}$, 50 variations are generated from X by uniformly scaling them by γ times the original length of X. Given $x_i \in X$, define a variation $x' \in X'$ computed by:

$$x'_i = x_{\lceil i/(\gamma) \rceil}, \tag{5.12}$$

where γ is a random value with $\gamma \geq 1$ and ranges to be given later for each experiment. This gives a dataset of 440 ($= (1 + 50) \times 40$) time series with 51 time series in every datasets. There are in total 40 datasets: $C_1, C_2, \ldots, C_{40}$.

Given this synthetic construction of the testing datasets, the correct classification and ranking results can be easily computed in advance. The correct classification result is that each of the 50 synthetic time series should be classified into the same set with its seed time series. The correct ranking result is that each of the 50 synthetic time series should be ranked according to the distance to its seed time series. Two metrics are used to evaluate the error of the search results, where e_c is called the *classification error* and e_r is called the *rank error*.

$$e_c = \frac{\text{the number of time series that should be in } C_i \text{ but are not}}{\text{the number of time series in } C_i}, \tag{5.13}$$

$$e_r = \frac{\text{the number of time series in } C_i \text{ that are in the wrong rank}}{\text{the number of time series in } C_i}. \tag{5.14}$$

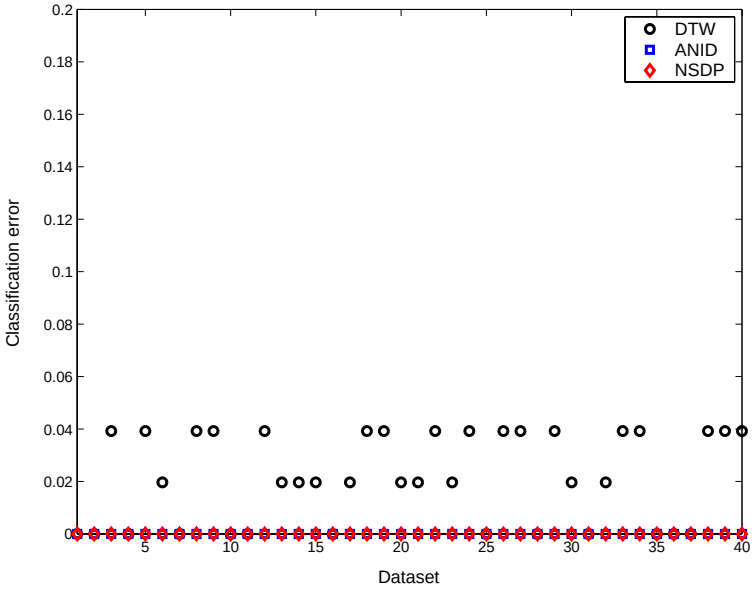
The rank is defined as the order in a sequence sorted according to the Euclidean distance metric in one classification.

The values of e_c and e_r of DTW, ANID, and NSDP are shown in Figure 5.4, when using different values of γ. The statistics of the errors are shown in Table 5.3. Figure 5.5, Figure 5.6 and Figure 5.7 show classification results returned by NSDP, ANID, and DTW, given two seed time series and two variations (two from each seed time series) that are computed from Equation 5.12.

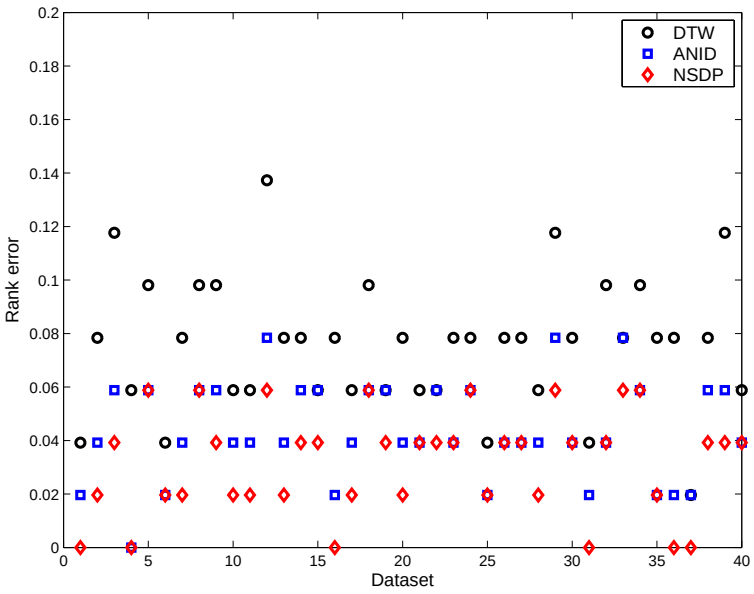
The experimental results show that DTW produces more both classification errors and sequence errors than ANID and NSDP when the differences of lengths of time series are large ($\gamma > 1.2$). When the differences of lengths become larger ($\gamma > 2.0$), NSDP produces fewer errors than ANID. This shows that NSDP has a greater tolerance to time scaling than DTW and ANID.

Table 5.3: Classification errors of DTW, ANID, and NSDP from applying uniform scaling.

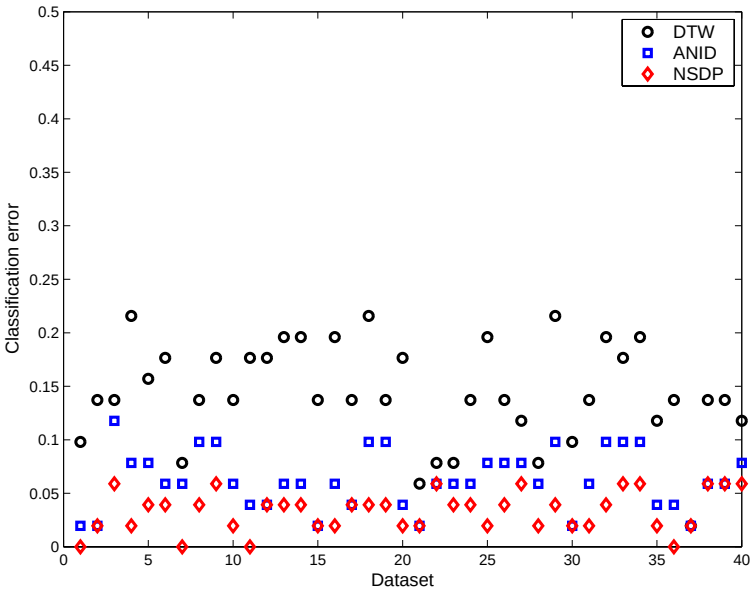
Approach	**Classification error e_c**					
	$1.2 < \gamma \leq 1.5$		$1.5 < \gamma \leq 2.0$		$2.0 < \gamma \leq 2.5$	
	Mean	Max	Mean	Max	Mean	Max
DTW	0.021569	0.039216	0.144118	0.215686	0.252941	0.411765
ANID	0	0	0.062255	0.117647	0.178922	0.274510
NSDP	0	0	0.033333	0.058824	0.154902	0.235294
Approach	**Sequence error e_r**					
	$1.2 < \gamma \leq 1.5$		$1.5 < \gamma \leq 2.0$		$2.0 < \gamma \leq 2.5$	
	Mean	Max	Mean	Max	Mean	Max
DTW	0.075490	0.137255	0.235784	0.372549	0.385784	0.568627
ANID	0.043627	0.078431	0.134314	0.215686	0.294608	0.470588
NSDP	0.031863	0.058824	0.094608	0.176471	0.256373	0.411765



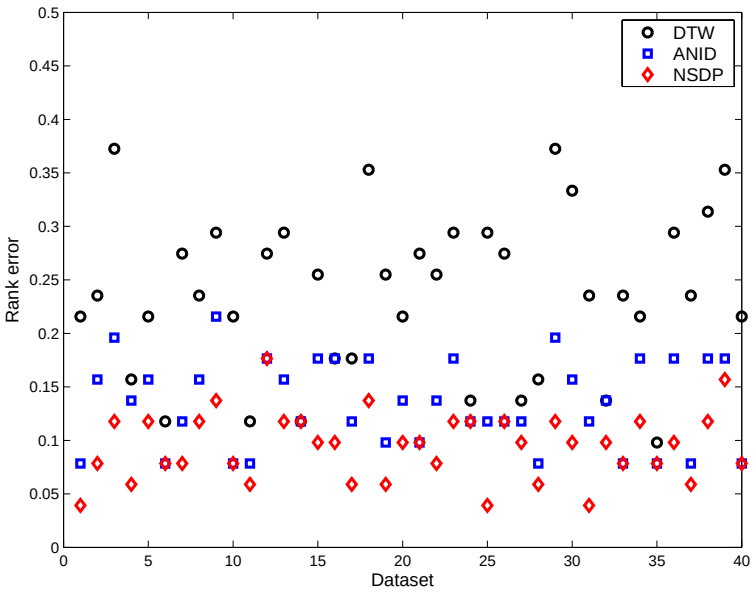
(a) Classification errors. $1.2 < \gamma \leq 1.5$.



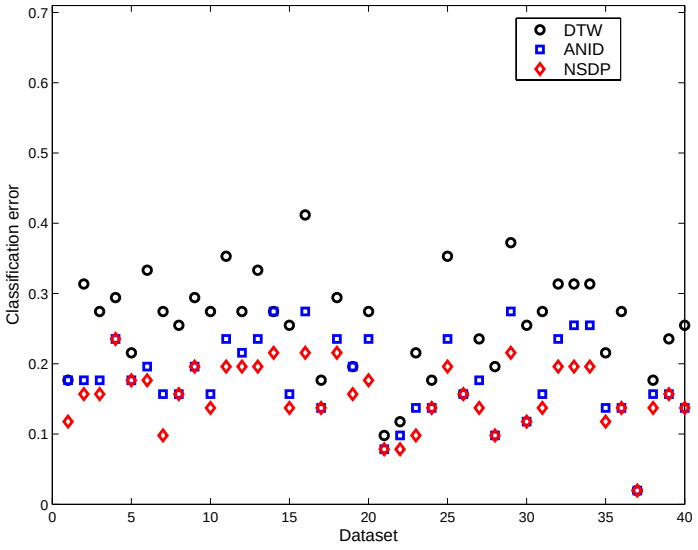
(b) Rank errors. $1.2 < \gamma \leq 1.5$



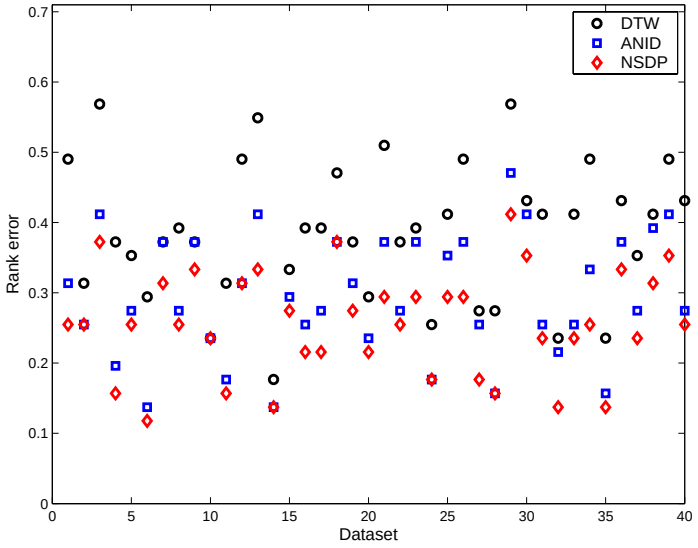
(c) Classification errors. $1.5 < \gamma \leq 2.0$.



(d) Rank errors. $1.5 < \gamma \leq 2.0$.

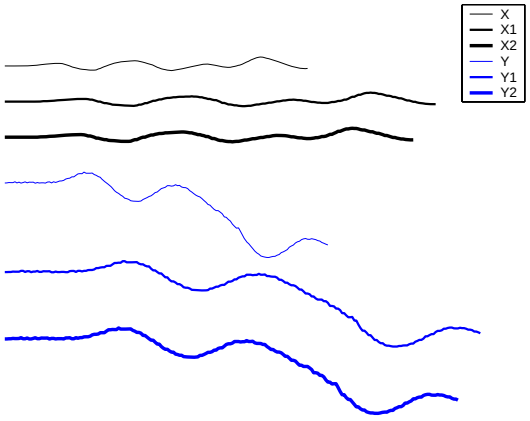


(e) Classification errors. $2.0 < \gamma \leq 2.5$.

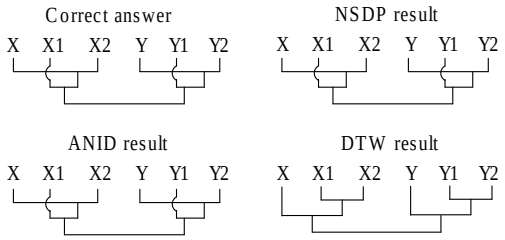


(f) Rank errors. $2.0 < \gamma \leq 2.5$.

Figure 5.4: Classification errors e_c and e_r produced by DTW, ANID and NSDP with different values of γ. The value e_c specifies the fraction of the number of time series that should be in a dataset C_i but are not over the number of time series in C_i. The value e_r specifies the fraction of the number of time series in a dataset C_i that are in wrong ranks relative to the number of time series in C_i.

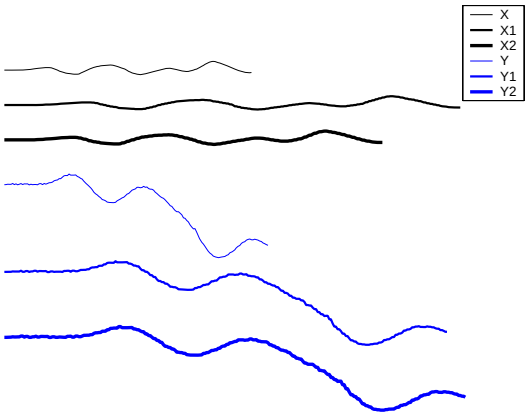


(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $1.2 \leq \gamma \leq 1.5$.

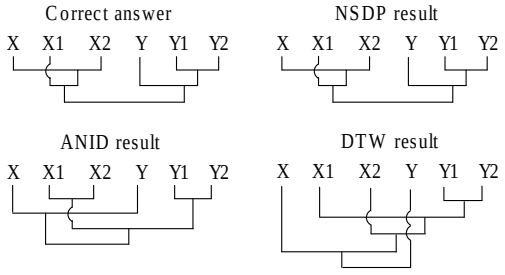


(b) Classification results. The result of DTW has rank errors.

Figure 5.5: Classification results returned by NSDP, ANID, and DTW when $1.2 < \gamma \leq 1.5$.

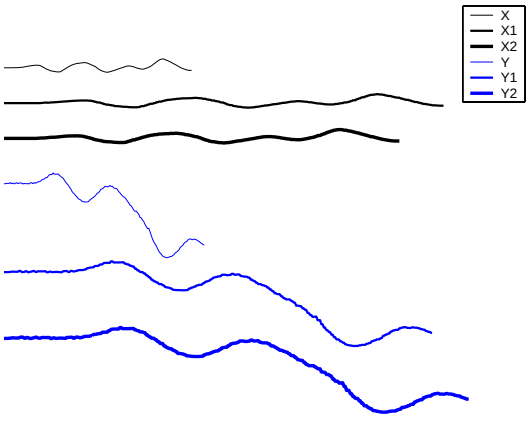


(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $1.5 < \gamma \leq 2.0$.

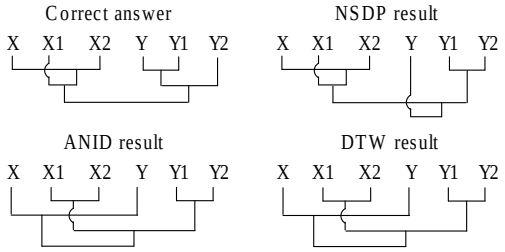


(b) Classification results. The results of ANID and DTW have classification errors and rank errors.

Figure 5.6: Classification results returned by NSDP, ANID, and DTW when $1.5 < \gamma \leq 2.0$.



(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $2.0 \leq \gamma \leq 2.5$.



(b) Classification results. The results of all approaches have classification errors and rank errors. But the result of NSDP is closer to the correct answer than that of ANID and DTW.

Figure 5.7: Classification results returned by NSDP, ANID, and DTW when $2.0 < \gamma \leq 2.5$.

Robustness to Additive Noise

For each time series $X \in \mathbb{D}$, 50 variations were generated from X by adding a factor μ of additive noise. Given $x_i \in X$, define a variation $x' \in X'$ computed by:

$$x'_i = x_i + \mu(\max(X) - \min(X)), \tag{5.15}$$

where μ is a uniform random value over various ranges to be defined.

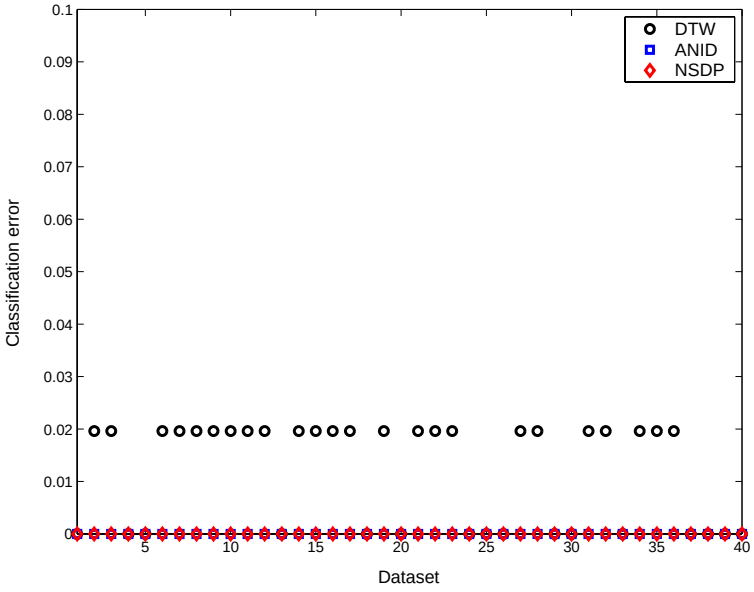
The values of e_c and e_r of DTW, ANID, and NSDP are shown in Figure 5.8, when using different values of μ. The statistics of the errors is shown in Table 5.4. Figure 5.9, Figure 5.10, and Figure 5.11 show classification results returned by NSDP, ANID, and DTW, given two seed time series and four variations (two from each seed time series) that are computed from Equation 5.15.

The experimental results show that DTW produces more both classification errors and sequence errors than ANID and NSDP when the additive noise values are large

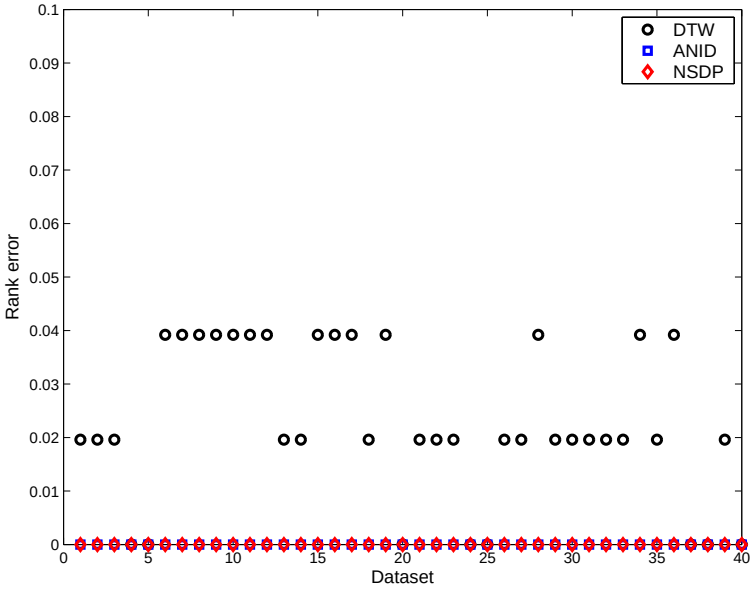
($|\mu| > 0.3$). When the additive noise values become larger ($|\mu| > 0.6$), NSDP produces fewer errors than ANID. This shows that NSDP has a greater tolerance to additive noise than either DTW or ANID.

Table 5.4: Classification errors of DTW, ANID, and NSDP from additive noise.

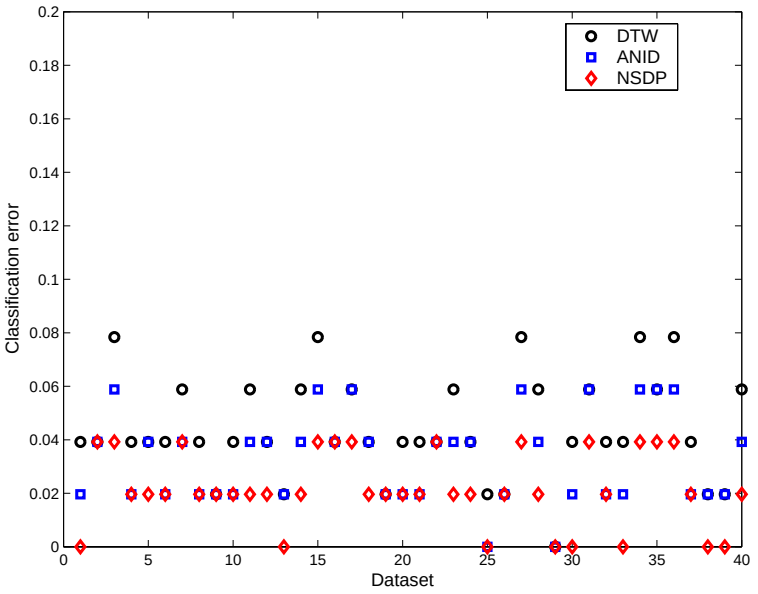
Approach	**Classification error** e_c					
	$\|\mu\| \leq 0.3$		$0.3 < \|\mu\| \leq 0.6$		$0.6 < \|\mu\| \leq 1.0$	
	Mean	Max	Mean	Max	Mean	Max
DTW	0.011756	0.019608	0.044118	0.078431	0.115196	0.196078
ANID	0	0	0.032843	0.058824	0.071569	0.117647
NSDP	0	0	0.021569	0.039216	0.051471	0.098039
Approach	**Sequence error** e_r					
	$\|\mu\| \leq 0.3$		$0.3 < \|\mu\| \leq 0.6$		$0.6 < \|\mu\| \leq 1.0$	
	Mean	Max	Mean	Max	Mean	Max
DTW	0.022549	0.039216	0.097549	0.196078	0.193627	0.294118
ANID	0.011765	0.019608	0.078431	0.156863	0.157843	0.235294
NSDP	0.011765	0.019608	0.058824	0.117647	0.139216	0.215686



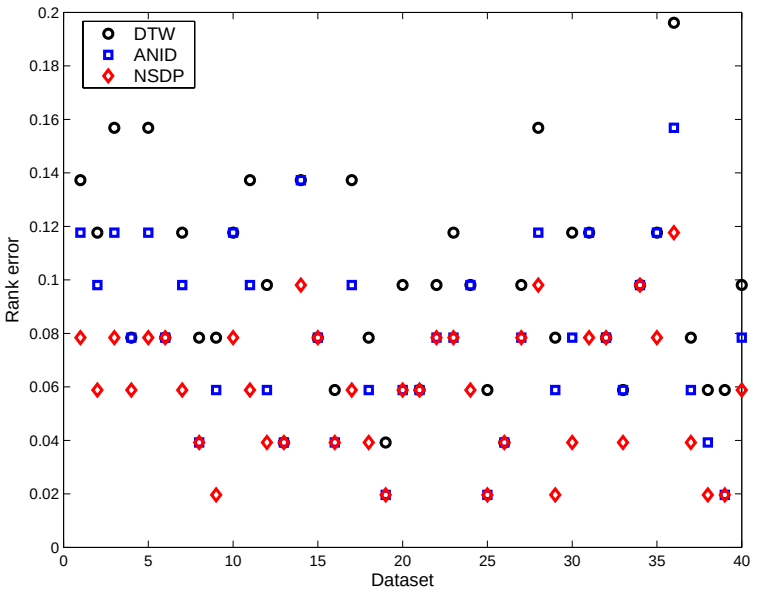
(a) Classification errors. $|\mu| \leq 0.3$.



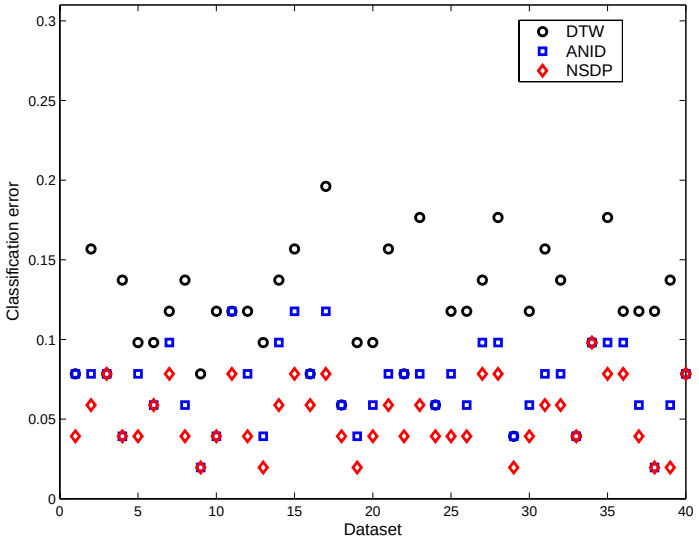
(b) Rank errors. $|\mu| \leq 0.3$



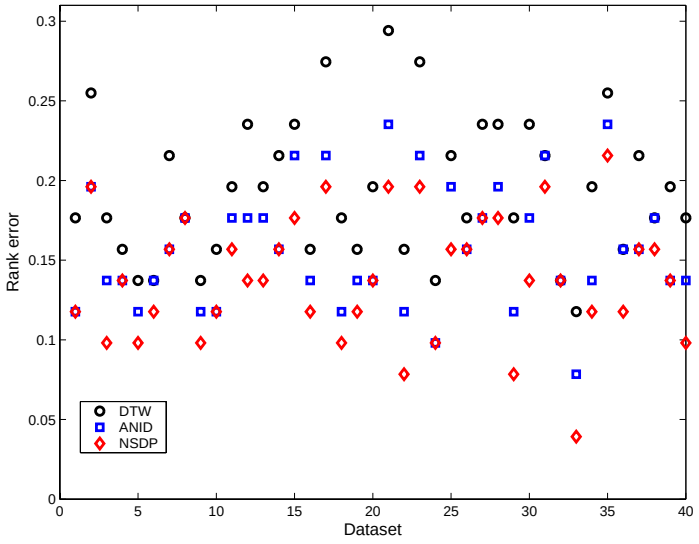
(c) Classification errors. $0.3 < |\mu| \leq 0.6$.



(d) Rank errors. $0.3 < |\mu| \leq 0.6$.

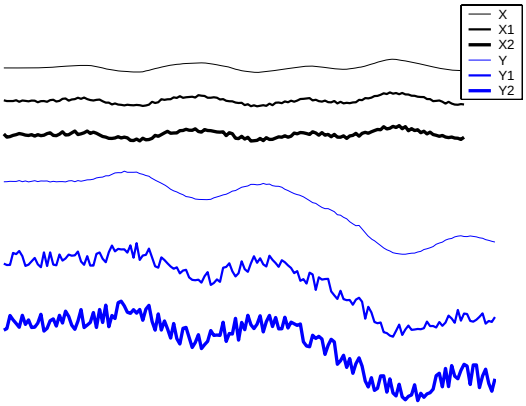


(e) Classification errors. $0.6 < |\mu| \leq 1.0$.

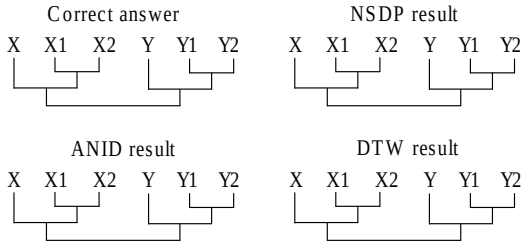


(f) Rank errors. $0.6 < |\mu| \leq 1.0$.

Figure 5.8: Classification errors e_c and e_r produced by DTW, ANID and NSDP with different values of μ. The value e_c specifies the fraction of the number of time series that should be in a dataset C_i but are not over the number of time series in C_i. The value e_r specifies the fraction of the number of time series in a dataset C_i are in wrong ranks relative to the number of time series in C_i.

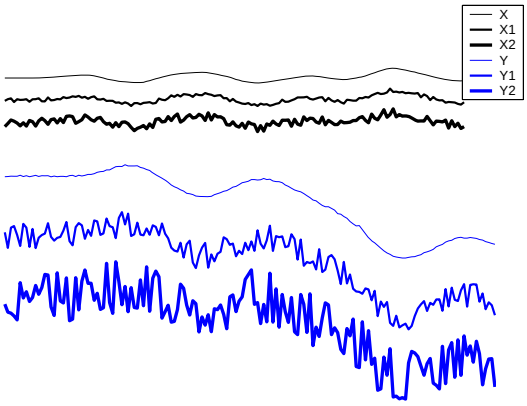


(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $|\mu| \leq 0.3$.

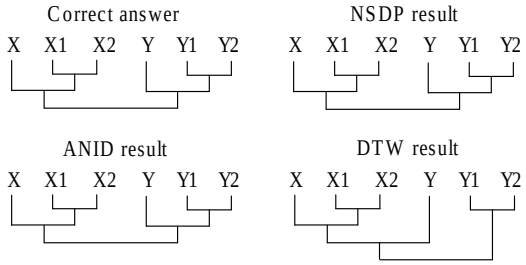


(b) Classification results. All results are correct.

Figure 5.9: Classification results returned by NSDP, ANID, and DTW when $|\mu| \leq 0.3$.

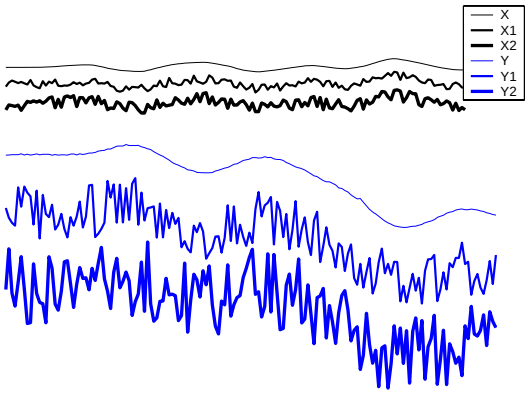


(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $0.3 < |\mu| \leq 0.6$.

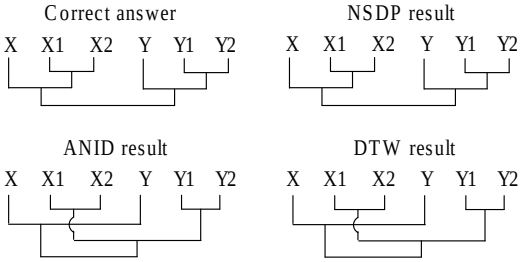


(b) Classification results. The result of DTW has classification errors and rank errors.

Figure 5.10: Classification results returned by NSDP, ANID, and DTW when $0.3 < |\mu| \leq 0.6$.



(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $0.6 < |\mu| \leq 1.0$.



(b) Classification results. The results of ANID and DTW have classification errors and rank errors.

Figure 5.11: Classification results returned by NSDP, ANID, and DTW when $0.6 < |\mu| \leq 1.0$.

Robustness to Impulsive Noise

For each time series $X \in \mathbb{D}$, 50 variations were generated by adding impulsive noise to some elements of X, i.e., for random $x_i \in X$,

$$x_i' = x_i + \rho(\max(X) - \min(X)), \tag{5.16}$$

where ρ is a random value with $\rho > 0$.

The values of e_c and e_r of DTW, ANID, and NSDP are shown in Figure 5.12, when using different values of ρ. The statistics of the errors are shown in Table 5.5. Figure 5.13 and Figure 5.14 show classification results returned by NSDP, ANID, and DTW, given two seed time series and four variations (two from each seed time series) computed from Equation 5.16.

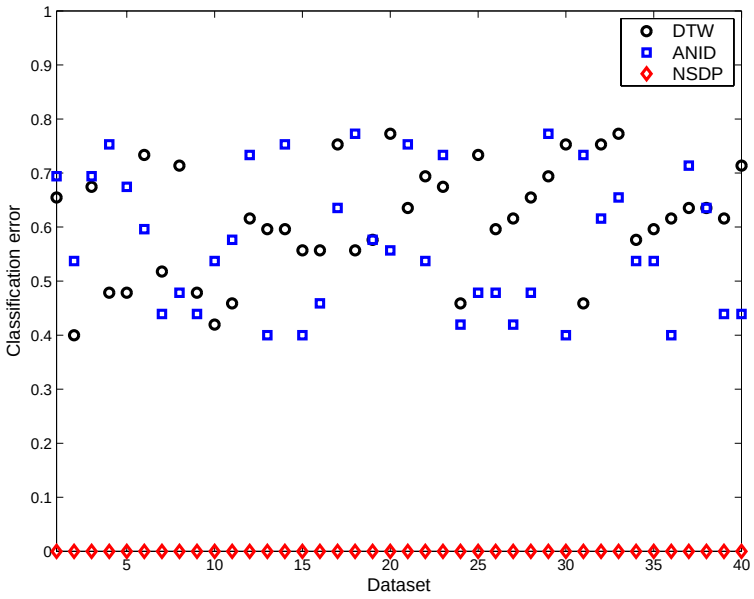
The experimental results show that DTW and ANID produce more both classification errors and more rank errors than NSDP when the data has impulsive noise. In fact, both DTW and ANID simply fail in the presence of such noise. In real data, such

outliners are common.

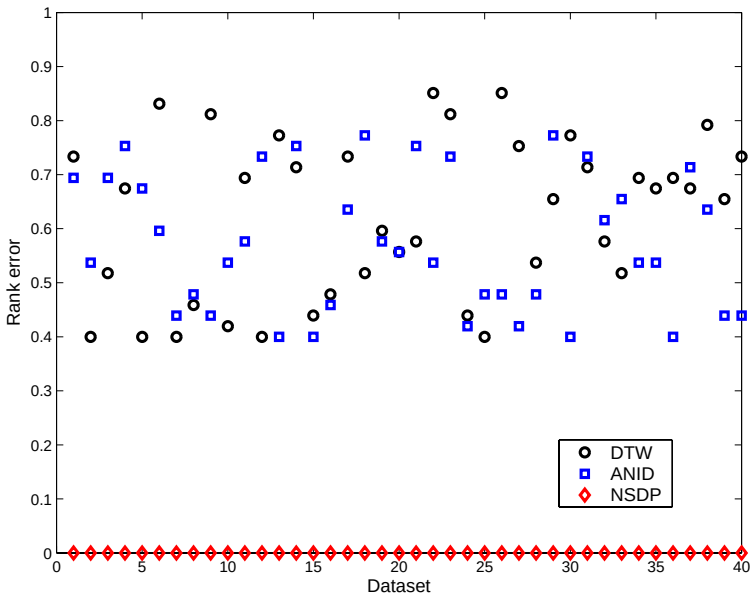
Table 5.5: Classification errors of DTW, ANID, and NSDP from adding impulsive noise.

Approach	**Classification error** e_c			
	$0.2 \leq \rho \leq 0.7$		$0.7 < \rho \leq 1.5$	
	Mean	Max	Mean	Max
DTW	0.611765	0.722549	0.619412	0.722549
ANID	0.572059	0.722549	0.571471	0.722549
NSDP	0	0	0	0

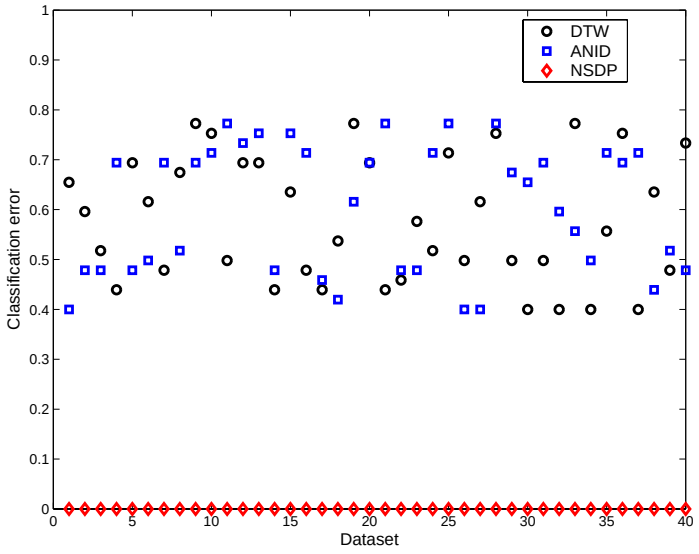
Approach	**Sequence error** e_r			
	$0.2 \leq \rho \leq 0.7$		$0.7 < \rho \leq 1.5$	
	Mean	Max	Mean	Max
DTW	0.623039	0.850980	0.631373	0.870588
ANID	0.611275	0.850980	0.618824	0.850980
NSDP	0.023529	0.039216	0.025157	0.039216



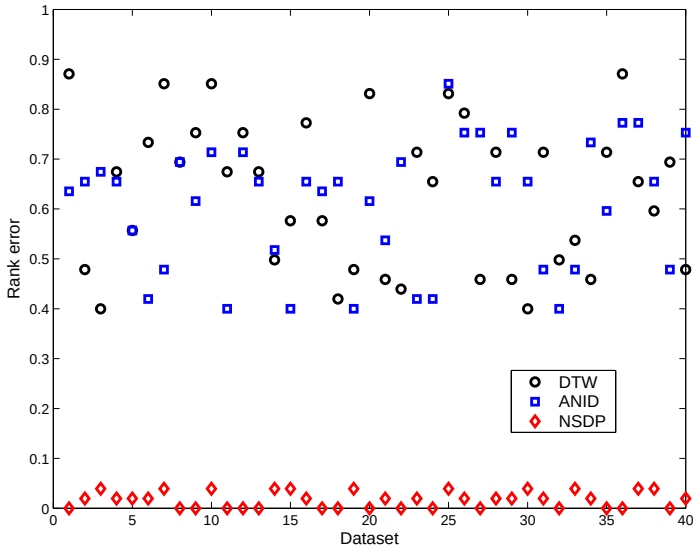
(a) Classification errors. $0.2 \leq \rho \leq 0.7$.



(b) Rank errors. $0.2 \leq \rho \leq 0.7$.

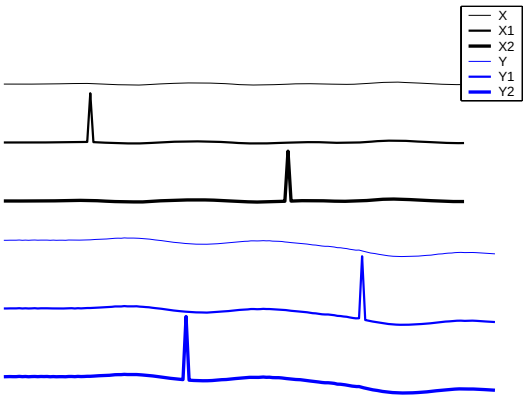


(c) Classification errors. $0.7 < \rho \leq 1.5$.

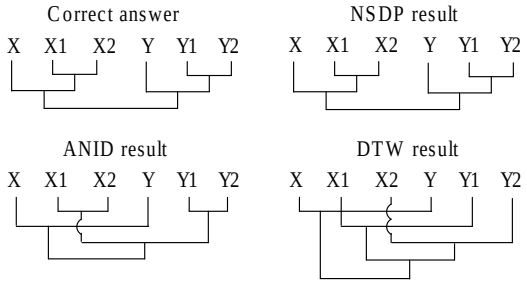


(d) Rank errors. $0.7 < \rho \leq 1.5$.

Figure 5.12: Classification errors e_c and e_r produced by DTW, ANID and NSDP with different values of ρ for impulsive noise. The value e_c specifies the fraction of the number of time series that should be in a dataset C_i but are not over the number of time series in C_i. The value e_r specifies the fraction of the number of time series in a dataset C_i are in wrong ranks over the number of time series in C_i.

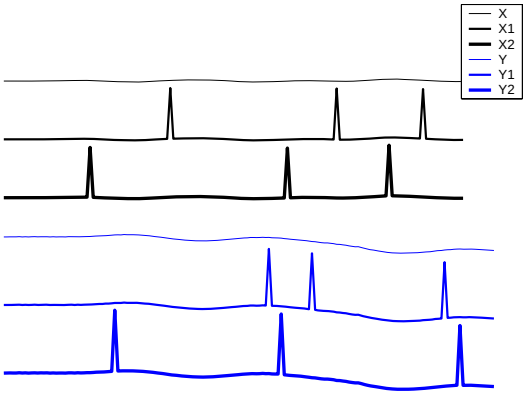


(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $0.2 \leq \rho \leq 0.7$.

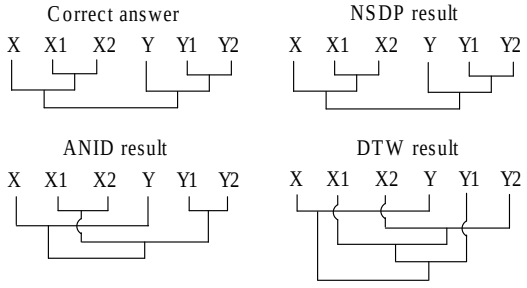


(b) Classification results. The results of ANID and DTW have classification errors and rank errors.

Figure 5.13: Classification results returned by NSDP, ANID, and DTW for impulsive noise when $0.2 \leq \rho \leq 0.7$.



(a) Time series X and Y and two variations of each $X1$, $X2$, $Y1$, and $Y2$ when $0.7 < \rho \leq 1.5$.



(b) Classification results. The results of ANID and DTW have classification errors and rank errors.

Figure 5.14: Classification results returned by NSDP, ANID, and DTW for impulsive noise when $0.7 < \rho \leq 1.5$.

5.4 Performance Evaluation

To evaluate the pruning power of the polynomial lower bounding method proposed in Section 3.2, several baseline lower bounding methods were selected for comparison, including Kim et al.'s method [KPC01], Yi et al.'s method [YJF98], and Keogh's method [Keo02]. All these methods were introduced in Section 2.4.2. To make the comparisons fair, all the lower bounding methods were used in the same C++ prototype system that implemented the NSDP approach. The prototype system ran on the same datasets for subseries matching.

The following parameter is used for the comparison of the pruning power of different lower bounding methods:

$$fp = \frac{\text{number of false positives}}{\text{total number of features}}. \tag{5.17}$$

The pruning power of different methods is shown in Figure 5.15. The values of fp are the average values over 50 queries. The smaller the values of fp, the better the pruning power. The total computational time of different methods is also show in Figure 5.16. The statistics of the results are shown in Table 5.6. The experimental results show that the proposed lower bounding method has both higher pruning power (i.e., produces fewer false positives) and because of this computing the final matches using NSDP requires less computational time overall than any of the baseline methods.

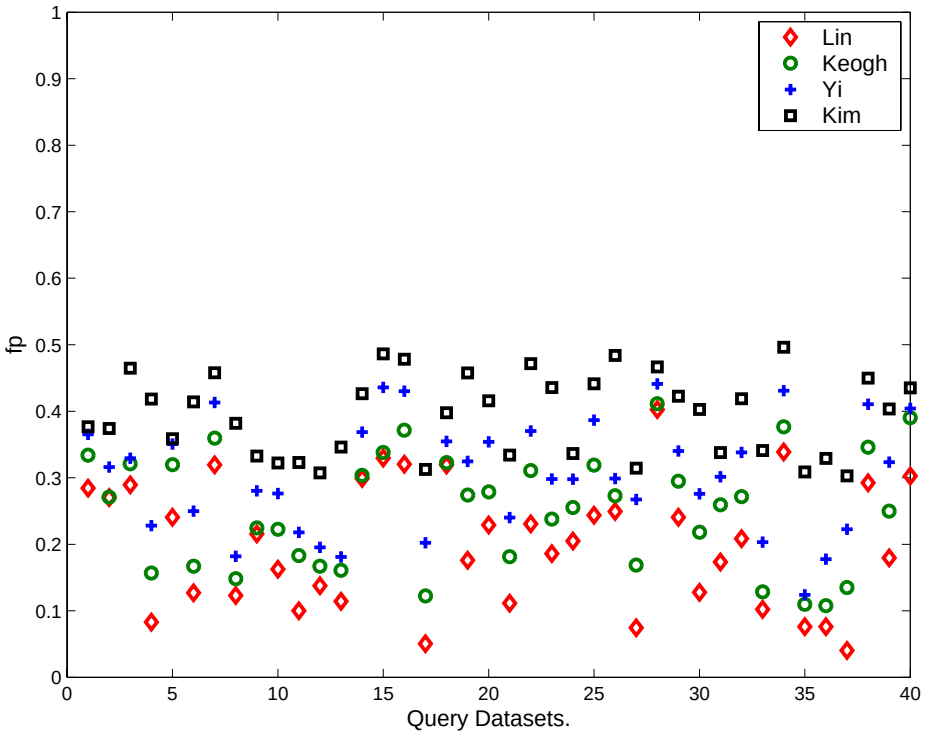


Figure 5.15: The pruning power of different lower bounding methods. The parameter of fp indicates the pruning power, which is defined as the fraction of the number of false positives over the total number of features. The fp values are the average values over 50 queries. The smaller the values of fp, the better the pruning power. The author's last name is used as the method's name, so Lin represents the results of the proposed lower bounding method used in NSDP.

Table 5.6: Statistics of the pruning power and overall computational time of different lower bounding methods. The author's last name is used as the method's name. So Lin represents the results of the proposed lower bounding method.

Approach	**fp**			**Time (Seconds)**		
	Mean	Max	Mean	Max	Mean	Max
Kim	0.3946	0.4961	0.3027	0.7054	0.7973	0.6039
Yi	0.3052	0.4410	0.1240	0.6485	0.7846	0.5472
Keogh	0.2523	0.4112	0.1076	0.5928	0.7207	0.4608
Lin	0.2013	0.4024	0.0403	0.4891	0.6221	0.3494

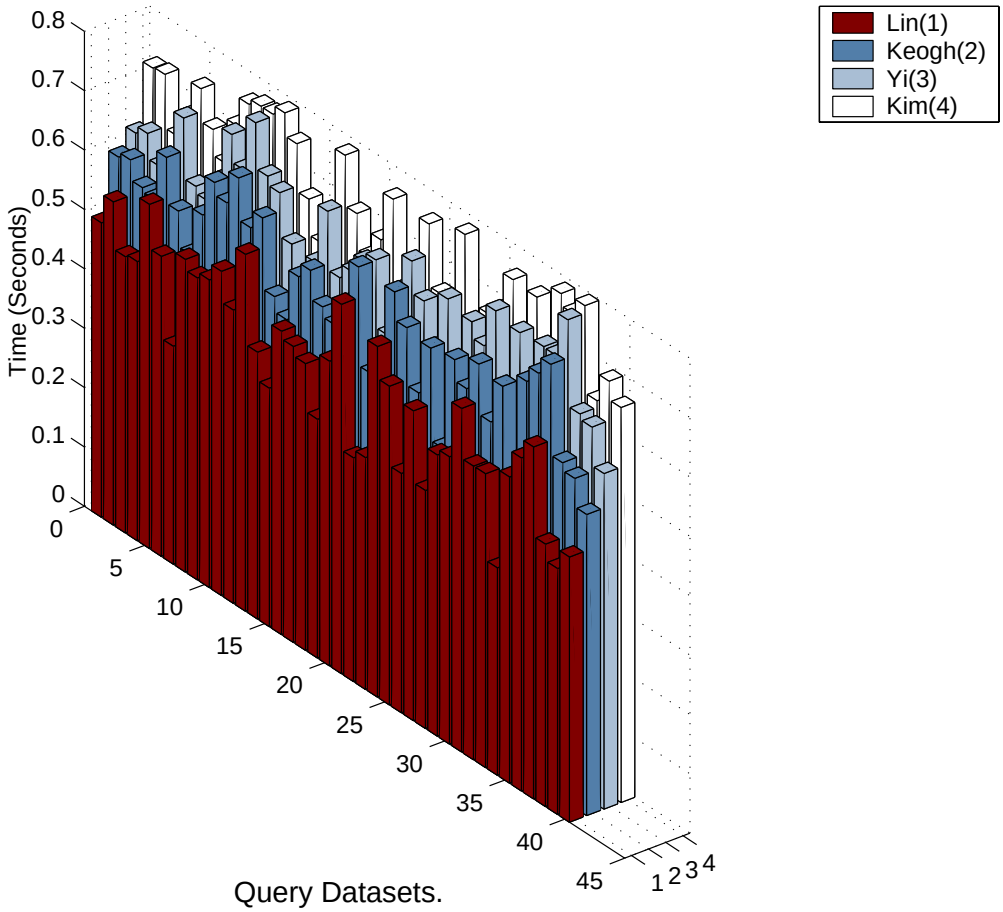


Figure 5.16: Overall computational time required by different lower bounding methods. The time values are the average values over 50 queries. The smaller the time values, the better the performance efficiency. The author's last name is used as the method's name, so Lin represents the results of the proposed lower bounding method used in NSDP.

5.5 Summary

In this chapter, the accuracy and efficiency of the proposed approach was evaluated using several baseline approaches for subseries matching and classification. However, since the sizes of the testing datasets are small, they can fit into main memory. This is not good enough for efficiency analysis, since in practice large databases need to be searched efficiently. Such large databases cannot fit into main memory. Therefore, in the following chapter, the performance of the proposed approach will be demonstrated on a larger database of motion capture data.

Chapter 6

Applications in Motion Capture Databases

To test the usefulness of the techniques discussed so far in an actual application area, in this chapter they are applied to various operations using a large motion capture database.

6.1 Introduction

Animated humans are an important part of computer animation, and they are commonplace in entertainment, training, and visualization applications. At present they are used as characters in games and for special effects in movies; they are part of simulations used by the military to prepare soldiers and by industry to instruct workers in using equipment; and they are used as visualization aids for medical analysis (studying an injured person's gait) and equipment design (determining if controls can be comfortably accessed). Moreover, there is every reason to believe that the demand for animated humans will grow in the future. Because much of our lives are spent observing and interacting with other people, animated humans are a natural and essential part of any visual medium designed to tell stories or simulate real world events.

However, producing animated humans is labor intensive. The most primitive method is *keyframed animation*, in which the animator specifies the important poses and lets the computer interpolate the in-between frames. This method originates from techniques of hand-made animation, in which humans, not computers, drew the in-betweens. Computers have saved people a lot of tedious work in animation production by automating significant portions of this process. However, realistic motions follow physical rules in the real world, and keyframing does not ensure these rules are obeyed. Also, keyframing is only as good as the animator's perception, understanding, and diligence in creating the desired motion.

These limitations have led to the idea of physical simulation, which generates motion following specified physical rules, such as Newton's Law. Compared with keyframing, physical simulation is a completely automatic method. However, this automaticity prohibits user interaction and makes motion editing difficult. In addition, its computational requirements are an obstacle for complex characters and high-performance or real-time projects.

It is hard for both keyframing and physical simulation to create natural human motions with realistic nuances. Fortunately, a third option is available. The swift development of hardware devices has enabled people to record the motion of a live person (or many other biological life forms). Sensors are placed on a person's body, and data describing the way an actual person moves can be sensed, digitized and stored in a computer. Alternatively, vision systems are being developed that can recover motion directly from images. These classes of techniques are called *motion capture*.

As its technology improves and its cost decreases, motion capture is attracting more and more research and commercial interest. Besides the use of motion capture to generate new animations directly, motion capture data can also be mapped onto different characters, with different shapes, a process called *motion retargeting*. This data can also be added to or mixed with keyframing or physical simulation to produce new motions, which is called *example-based synthesis*. Of course, there are more ways

to use motion capture data than those listed here.

Unfortunately, digitized motion data is expensive to create and manipulate. Its creation requires either the talents of a skilled animator using specialized software, or finicky motion capture hardware that may produce noisy data, or both. Also, some motions are hard to capture given the limits of the hardware or may require the expensive cooperation of an actor or athlete. The motion data that goes into the production of a feature animation, game, or other similar projects represents an investment of millions of dollars. As a studio accumulates more and more such data, it is in its best interest to leverage this investment.

Recently, large motion capture databases have become commonplace due to real-world projects requiring expressive character motions. These databases contain many different kinds of actions and any given kind of action can have many variants. Theoretically, it seems that we do not need to capture motions redundantly and that we could create realistic motions simply by connecting the appropriate motions (or sub-motions) in the database [LWS02]. This is feasible only if users can find appropriate motions fast enough. To do so, we need an efficient way to search and cluster the data.

Motion capture data are multi-channel time series. Therefore, searching for motion of the same style as an example motion is actually a problem of matching for time series, and clustering motions of the same style is actually a problem of subseries join for time series. To apply the generic results in time series indexing and compression presented in previous chapters to motion capture data, a few domain-specific extensions should be included. These extensions include both dealing with multiple channels, but also some adjustments for dealing with perceptual effects specific to animation.

6.2 Motion Capture Data Representation

To represent motion capture data, a *skeleton* representation is combined with sequential pose data for each *degree of freedom (DOF)*. A skeleton is a tree-like structure that

records the structure of the rigid bones of the character. A weight is assumed to be available at each joint proportional to its importance. One DOF gives the root position and the other DOFs typically give joint angles. In motion capture databases, commonly-used file formats for motion capture data include BVH, ASF, and AMC.

A character motion cannot be represented without a character skeleton since the parameters in the time series are given relative to it. Different skeletons have different motion representations and parameterizations. Section 6.2.1 will introduce some techniques used to represent the human skeleton and relate it to the time series representation. A motion is a chain of frames (or poses) in a time sequence. Section 6.2.2 will describe what parameters are used to represent a frame. This chapter will focus on human animation specifically, although the results can be extended to motion capture databases containing data for of other kinds of skeletal structures.

6.2.1 Representing Skeletons

A skeleton is a collection of bones that are connected in a specific way. A human skeleton is a tree structure (see Figure 6.1). Generally, in motion capture databases, a skeleton representation only records the configurations of the rigid bones of the character.

There are two ways to represent a skeleton. One is the non-hierarchical representation. Each bone's configuration is independent from others. The advantage of this representation is that changing the configuration of one bone will not influence those of others. However, with this approach preserving the connectivity between bones requires extra constraints. This drawback makes most people use another representation—the hierarchical representation. In a hierarchical representation the motion parameters of each bone relative to its upper-level bone (its parent node in the tree) are recorded. Changing one parameter will not destroy the connectivity between bones. This representation also has problems: it can suffer from interlock due to the difficulties in representing rotation angles and it is not intuitive to directly specify the positions of

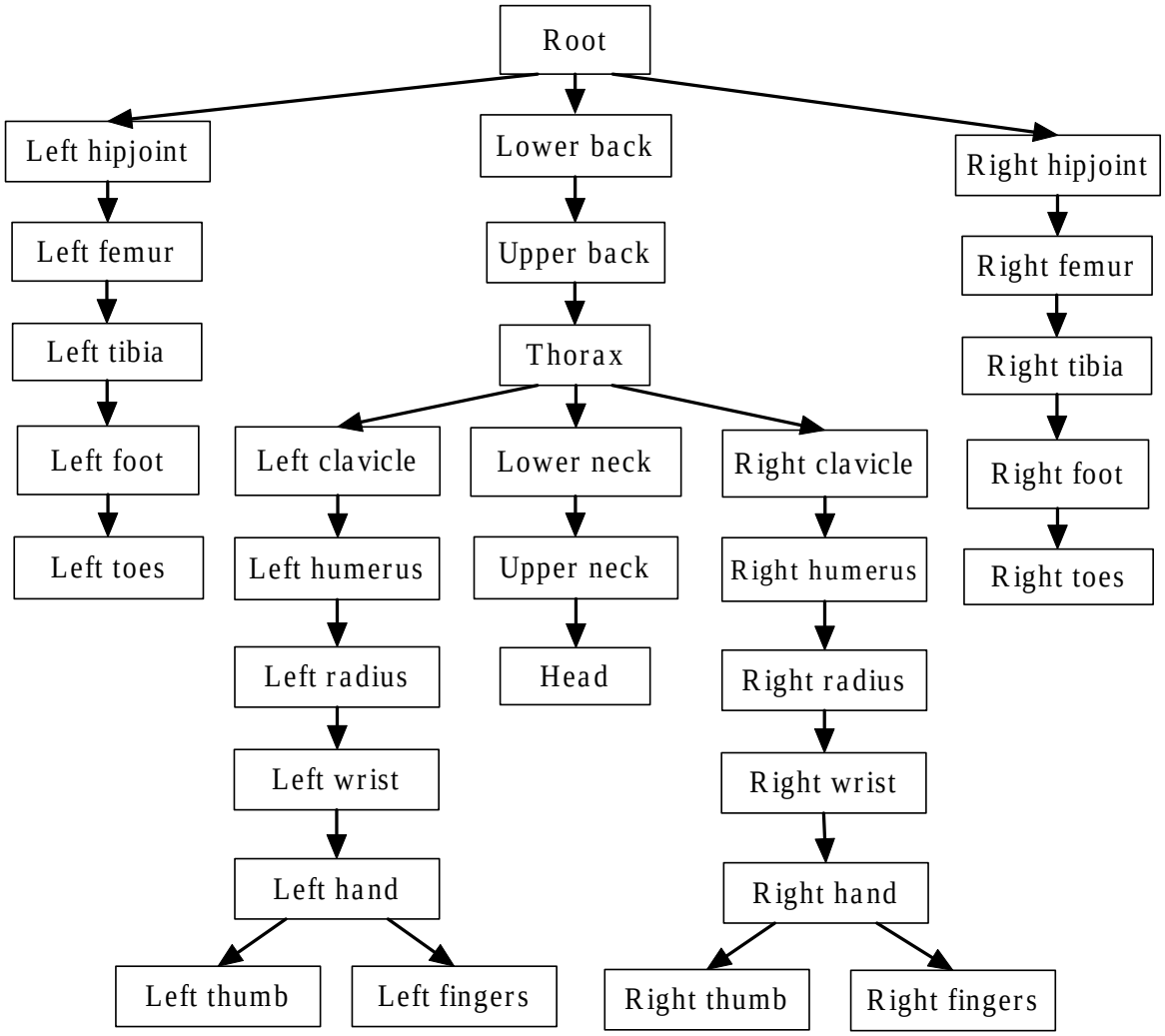


Figure 6.1: A typical hierarchical structure of a simple human skeleton from an ASF file.

endpoints such as hands and feet.

6.2.2 Representing Frames

A *frame* of a motion records the body position and orientation and the orientations of all bones. The body position is usually represented as coordinates in a world coordinate system. The world coordinate system is a global Cartesian reference space. The body orientation is a rotation with respect to $X-Y-Z$ axes in the world coordinate system. A bone orientation is a rotation relative to its parent's local coordinate system.

There are several ways to represent a 3D rotation, such as Euler angles, quaternions, and rotation matrices. Each of them has advantages and disadvantages. Current common motion file formats, such as BVH or AMC, use Euler angles to represent orientation. Euler angles use three real numbers to represent a sequence of three individual rotations around the coordinate axes. Although these numbers are not independent, i.e., changing one of the angles alters the meaning of the subsequent angles, this disadvantage does not matter to motion retrieval and clustering, since motion data for searching is not modified. However, for the search to be meaningful a representation is needed for both the query and the data using a common set of conventions.

One complication relative to the previous work discussed in this thesis is that motion capture data often includes rotations in 3D space, which can be difficult to parameterize in a way that makes computing the distance between them uniform. Computing the Euclidean distance between Euler angles, in particular, is *not* uniform. Therefore, the distances between bone endpoints are used in my prototype system. These are in turn computed from Euler angles.

6.2.3 Pose Similarity

Given two poses (frames):

$$\begin{aligned} m_i &= (m_{i,1}, m_{i,2}, \ldots, m_{i,k}) \\ m_j &= (m_{j,1}, m_{j,2}, \ldots, m_{j,k}), \end{aligned}$$

a typical distance metric used for pose (frame) comparison is the weighted Euclidean distance:

$$d_p(m_i - m_j) = \sqrt{\sum_{r=1}^{c} w_r (m_{i,r} - m_{j,r})^2}, \tag{6.1}$$

where c is the degrees of freedom and w_r is the r^{th} weight for the degree of freedom.

There are some variations to Equation 6.1, for example see Lee et al. [LCR+02]. However, Equation 6.1 has become the basic universal distance metric of frame comparison for motion capture data. What is not agreed on is how the weights w_i should be set. It is obvious that some bones are more important than others, but how to specify or derive the weights is still under discussion.

For example, Zordan et al. [ZMCF05] have assigned high weights to the trunk parts and lower ones for the limbs. However, to avoid sliding ground contact, they also computed the center of mass and assigned high weights to the closest ground-support body.

Wang and Bodenheimer [WB03] further optimized the weights in the metric function given by Lee et al. [LCR+02]'s metric function. They took a set of different motion segments, which consist of a variety of motions including walking, running, jogging, dancing, and gesturing, in different styles. Their distance metric computes transition points between segments. The optimal transition points are selected by experienced animators. They defined a good transition as one showing invisible visual discontinuity and a bad transition as one showing obvious visual discontinuity. Then they optimized weights using a constrained least-squares minimization: $\min_w ||Dw - o||_2^2$, where w is

a vector of weights, D is a matrix of the similarity distances of Equation 6.1, and o is a vector of ones and zeros—the human-made optimal transition vector. The optimization is constrained in a way that the weights are non-negative and body-symmetric, for example, the weight for the left shoulder should normally be identical to the right shoulder.

The drawback of a distance function such as a L_p-norm based on the entire set of data is its low efficiency. For example a full body skeleton usually has about 30 bones, each bone has 3 rotation parameters, and plus 6 body move parameters (3 for translation and 3 for rotation) there are about 100 parameters for each frame. This is the amount in data of one frame recorded in common motion file formats, such as AMC [Acc].

Numerical comparisons of motion capture data using a L_p-norm distance function also may not reflect perceptual similarity. In practice, a set of found motions can be used as new queries to find similar motions [KG04]. In this way perceptually similar motions can be found iteratively. Obviously, this method requires more computation time and often user interaction to select the best matches at each step.

6.3 Related Work

This section introduces related work on retrieval and compression in the areas of motion capture and animation.

6.3.1 Retrieval of Motion Capture Data

Motions can be retrieved on two semantic levels. One is query by a textual metadata description. Most existing motion capture databases depend on human generated annotations or decisions, such as labeled motions with texts of “walk”, “run”, or “dance”, and these can of course be retrieved by a textual query. The other approach is query-by-example, which uses a short motion (or a sub-motion), to retrieve all motions in the

database containing parts or aspects similar to the query. Most current prototypical systems (and the prototype system for this work) target the latter query mode.

The typical distance function for comparing motion capture data is the Euclidean distance given by the sum of the distances between each bone parameter. The distance functions might vary with different applications, as do the actual values that are compared. For example, Arikan and Forsyth [AF02] defined *frame distance* as a weighted sum of the Euclidean distance between joint positions and velocities, measured relative to the root's coordinate frame. Lee et al. [LCR+02] defined the distance between frames as a weighted sum of the great-arc distance between joint orientations and the Euclidean distance between joints. In the literature, joint data is usually either represented in the global coordinate frame (to preserve interactions with stationary objects) or relative to the root's local coordinate frame. Usually, a weight parameter is also given to each bone to specify influence of the bone on the whole pose. As we have mentioned, Wang and Bodenheimer [WB03] optimized the weights based on the cost metric used in Lee et al.'s work [LCR+02].

Motion capture data is an example of multi-channel time series data. The typical indexing methods either follow the GEMINI framework as in Cardle et al. [CVB+03], or build similarity graphs (or matrices). The similarity graphs (or matrices) build a compact representation of all possibly similar motion segments by comparing each pair of motions. To automatically construct transitions, several recent research efforts have identified locally similar regions in a motion capture dataset [AFO03, KGP02, WB03].

Liu et al. [LZWP03] automatically extracted keyframes for each motion in a database and used these keyframes to construct a hierarchical tree of clusters of motions, with deeper levels of the tree corresponding to joints deeper in the skeletal hierarchy. To process a query, the closest leaf cluster is found and its motions are directly compared against the query. This algorithm also uses a direct numerical comparison to determine similarity, and it is designed to compare entire motions against a query. For other motions, such as rhythmic motion, Kim et al. [KPS03] automatically identified similar

motions using beat analysis to segment a motion dataset and then clustering motions based upon a similarity metric. The approach proposed in this thesis applies to more general datasets and is geared toward content-based search and clustering.

Recently, researchers begin to consider derived discrete geometric features as distance metrics directly. One example is Muller et al. [MRG05] who presented a system in which such user-specified geometric features are a part of the query with the motion itself. Their indexing strategy is also based on these features. One drawback of this system is that it has to reindex the whole database from scratch whenever the features used for query are modified. It also needs the user to do more work compared with other systems. To overcome of drawbacks of Muller et al.'s approach, the author's previous work [Lin06] used a group of general geometric features, which reflect the relationships between perceptually important joints, to index the database. This method does not require the user to input the geometric features. However, the accuracy of the method is sensitive to joint selection and the method is not guaranteed to find the best matches. Although these drawbacks make geometric feature based methods unsuitable for searching large databases, these methods do provide the user with a high degree of flexibility.

For large databases, a brute-force search that examines every frame in the database sequentially demonstrates extremely poor performance. Many indexing strategies have therefore been developed to partition the database and/or to cluster motion segments into hierarchical structures. There also exist several methods for motion segmentation. Probabilistic PCA, as an extension of the classic PCA surveyed in Section 2.3.2, models the residual variance discarded by PCA [TB99]. Safonova et al. used the probabilistic PCA method to divide a motion sequence into segments of distinct behaviors [SHP04].

Switching Linear Dynamic (SLD) models are used in human motion synthesis, classification, and visual tracking. Since exact inference in the SLD model is intractable, approximate algorithms are usually used. Pavlović et al. proposed a variational inference algorithm which casts the SLD model as a Dynamic Bayesian Network [PRM00].

Li et al. modeled a *motion texton* using a linear dynamic system and represented the texton distribution by a transition matrix indicating the likelihood of transitions between textons [LWS02].

6.3.2 Compression of Motion Capture Data

Motion database compression exploits redundant data across three dimensions: the temporal dimension, the DOF dimension, and the motion clip dimension [Ari06]. Most compression approaches use a decorrelation step followed by a coding step. Decorrelation removes the redundancy between different data elements and coding exploits the differences in probability between different data values to reduce the data rate. High-probability data values can be coded with shorter codes than low-probability data values, reducing the average bit rate. Coding algorithms are relatively standard; most compression schemes differ primarily in their approach to decorrelation, since different data types have different kinds of redundancy.

Decorrelation of Temporal Redundancy

For motion capture data, decorrelation of temporal redundancy can be done by change of basis, and wavelet transformations are especially useful. Guskov and Andrei [GK04] encoded differential wavelet coefficients to compress an animation sequence. Beaudoin et al. proposed a modified wavelet technique [BPvdP07] with properties well-suited to motion data. They worked directly with joint angles and used a cubic interpolating spline wavelet basis.

Liu and McMillan [LM06] segmented the motion sequences using the probabilistic PCA method. Temporal redundancy is then exploited by adaptively fitting cubic splines to the reduced-basis coefficients and only storing the keyframes for the resulting cubic splines.

Other temporal simplifications have been used to extract key poses in an animation sequence [ACCO05, KM04], space-time optimization [LGC94], and motion edit-

ing [LS00, LS99]. Ibarria and Rossignac [IR03] proposed a predictor/corrector method to exploit temporal coherence between frame meshes.

Decorrelation of Redundancy between Degrees of Freedom

Compression can also exploit redundancy between DOFs. Often the behavior of a large number of DOFs can be expressed relative to the behavior of a lower-dimensional set of data. Safonova et al. [SHP04] used a low-dimensional space to represent high-dimensional dynamic human behaviors. Their work showed that 10 to 20 DOFs can accurately represent a motion of 40 to 60 DOFs for a typical human skeleton model. Representations of poses in a reduced dimensional space have been proposed for applications other than compression, including animation retrieval [FF05], motion editing [BSP+04, GMHP04], and motion synthesis and texturing [CH05, GBT04, PB02, RCB98].

PCA can also compress motions by exploiting inter-DOF redundancy. Liu and McMillan [LM06] used PCA to extract a reduced marker set that can represent a full body pose. PCA can also be used to compress meshing shapes. PCA compresses shapes by finding portions of the mesh that move rigidly and only encoding the rigid transformation and residuals [Len99, GSK87].

Compressing in both temporal and DOF spaces can achieve better compression rate than temporal compression only [Ari06]. However, compressing individual DOFs makes reuse of the motion data and update of the database easier.

Decorrelation of Redundancy Between Motion Clips

Compression over the motion clip dimension is useful when the database has many related motion clips. Arikan [Ari06] applied clustered PCA to compress linearly related motion clips. They connected all motion clips in the database into a long sequence. They uniformly divided this sequence into segments of same length, then exploited both joint correlations and time coherence by using PCA for each segment. Instead of

using joint angles directly, virtual markers computed from joint angles are used as an internal representation in their work. Since joint angles are required by current game and simulation engines, extra time and storage are needed for conversion back from this representation, however.

This method has a good compression rate but requires a complicated and expensive compression procedure. If the database contains too many linearly unrelated motions, which is a common case, then clustered PCA among motion clips may also produce artifacts. The efficiency of this method also depends on the settings of some heuristic parameters. Their uniform segmentation strategy might also require recompression of the entire database when it is updated.

Relative to previous work, the proposed compression approach is most comparable with the wavelet approaches, since only temporal redundancy is exploited. However, this has advantages since it makes database update and access easier. To compare with methods suitable to similar use cases, the performance of the compression approach for motion capture data proposed in Section 3.3 will be compared with the best previous wavelet approach [BPvdP07] as well as with a Haar wavelet approach, in Section 6.6.2.

6.4 Search and Join of Motion Capture Data

A motion can be regarded as a c-channel time series of length n. The value c specifies the number of degrees of freedom, including the translation and rotation of the root and rotations for each joint. The value n specifies the number of frames. The motion is assumed to be sampled at regular intervals and the number of DOFs does not change from frame to frame or between motion clips. The data for each DOF can be modeled as a series $M = (m_1, m_2, \ldots, m_n)$, which can be interpreted as a sampled curve. The anisotropic diffusion analysis and the non-uniform segmentation method proposed in Section 3.3 can be applied to break this curve into variable-length segments at its own natural discontinuities. Motion databases often have a great deal of noise. Fortunately,

the anisotropic diffusion scale-space analysis and the proposed subseries join approach can effectively deal with this noise, as will be shown.

The skeleton structure of the character indicates the importance of the bones. For most motions, only some bones dominate the pose, such as the back, the arms and the legs. In my prototype system the weights are used to reflect this in the distance function. The system also allows the user to select the bones of interest and ignore other bones. The bones of interest are called the *featured joints*. The featured joints in the proposed implementation are the trunk (combining the joints of the lower back, the upper back and the thorax), the left and right upper arms (combining the joints of the clavicle and the humerus), the left and right lower arms, the left and right femurs, and the left and right tibias. These bones are chosen because they influence the visual similarity most. In this way, the number of channels can be reduced. An application can easily select feature bones and weights based on the user's input. It would also be possible to use PCA [FF05] to reduce the dimensionality of the data.

Using the feature representation introduced in Section 3.2, a hierarchical feature structure is created for each channel of a motion time series. The approach to search and cluster motion capture data uses the approach introduced in Chapter 4, except that motion capture data are multi-channel time series. The result of motion subseries is the minimum subseries join that contains all matching subseries of all channels of the featured bones. Note that this "merging" method used in my implementation works well for searching as will be shown in Section 6.6, but it is not a good alignment method for multi-channel alignments. More discussions will be given in the conclusion of this chapter. If we assume s_i is score of the i^{th} channel returned by the similarity measure and w_i is the weight value assigned for this channel, then the distance is $\sum_{i=1}^{c} w_i s_i$, where c is the number of channels.

My prototype system can flexibly deal with the tradeoff between accuracy, efficiency, and memory usage by choosing feature bones and selecting scales of the anisotropic diffusion process. When there are fewer channels of interest, finer scales of the index

can be used for better feature matching.

6.5 Compression of Motion Capture Data

This section shows how to apply the compression approach introduced in Section 3.3 to motion capture data. Motion data at each DOF is first divided into segments at feature discontinuities. Every segment at each DOF is approximated by a cubic Bézier spline. the control points of cubic Bézier splines are hierarchically differenced. An arithmetic coding algorithm is finally used to further encode the differences. Decompression is the inverse process of compression.

For human motion, the major post-processing step deals with the *footskate* problem. Footskate occurs when a character's foot slides on the ground when it should be planted firmly. As mentioned in [BPvdP07], visible artifacts will appear when the compression error is larger than a certain percentage. A motion capture compression scheme has to minimize footskate. There exist sophisticated methods to solve the footskate problem [KGP02, Ari06, BPvdP07]. The method proposed in [BPvdP07] is used in the my prototype system, which compresses the foot joints separately with a tighter error tolerance than the remaining data and uses inverse kinematics to correct the motion of other joints.

Dealing with the footskate problem may not be necessary for all applications. For example, a game engine synthesizes generated keyframes to combine them with motion capture data and a motion smoothing process will be performed. Compressing the foot joints separately with greater accuracy does not influence the overall compression performance greatly [BPvdP07].

6.6 Experimental Results

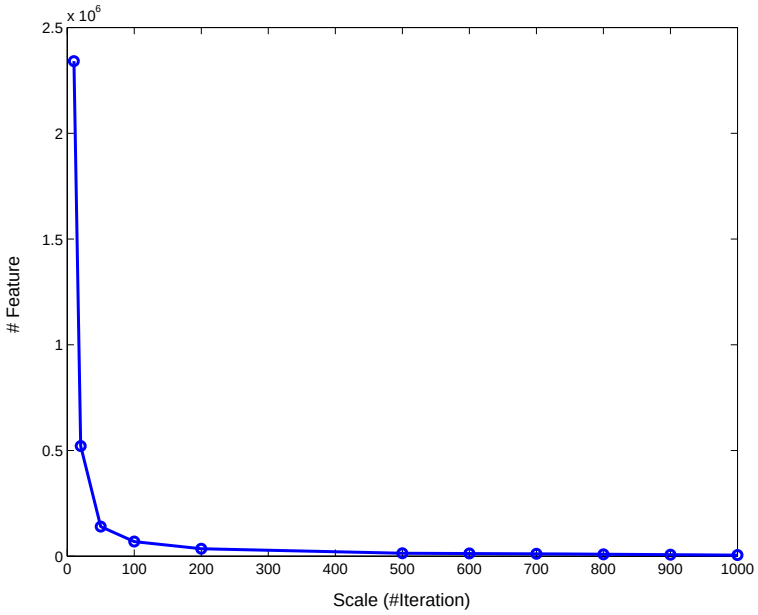
My prototype system was tested on a 3.15GB motion capture database [Gra] containing 4 million frames (about 18.61 hours sampled at 60Hz). This database contains 2493 AMC files ranging in length from about 300 to 23000 frames. The skeleton used for this data has 30 degrees of freedom (channels of featured joints). All the experiments were run on Linux Kernel 2.6 PC with 512MB RAM and Pentium IV 3.0GHz CPU. The database contains various kinds of motions, including walking, running, kicking, jumping, boxing, dancing, and gymnastics. The total and average numbers of features generated at different scales by my prototype system are shown in Figure 6.2.

6.6.1 Subseries Matching and Join

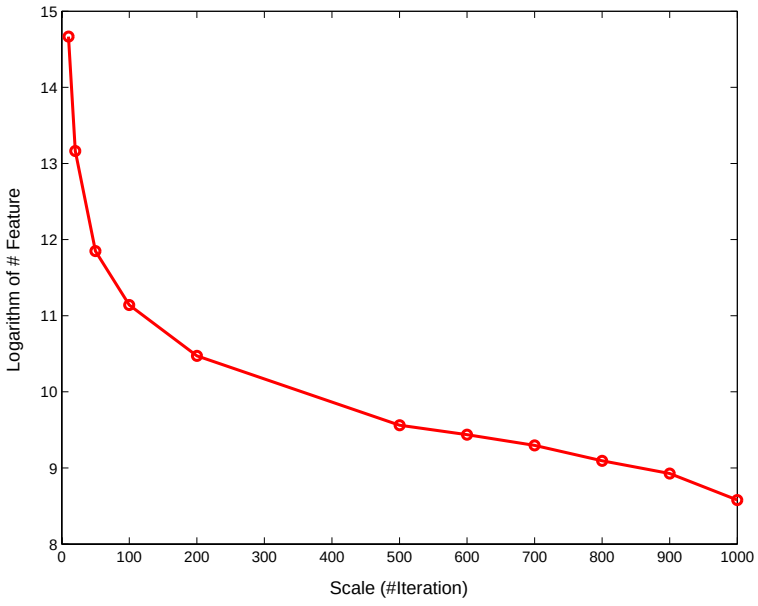
My prototype system needs about 3.5 hours for index construction for the whole database when the finest scale $\tau = 500$ is manually selected. Indexing is a pre-processing step and does not influence the retrieval speed. Also, it is possible to add new sequences to the database without recomputing the index from scratch, so it can be built incrementally.

To test searching performance, 100 random motion subseries were selected as test queries out of the database with lengths ranging from 59 to 376 frames. During the test searches, the queries are removed from the database (if they are left in, they are always found as their own best match). Table 6.1 shows performance of my prototype system for subseries matching.

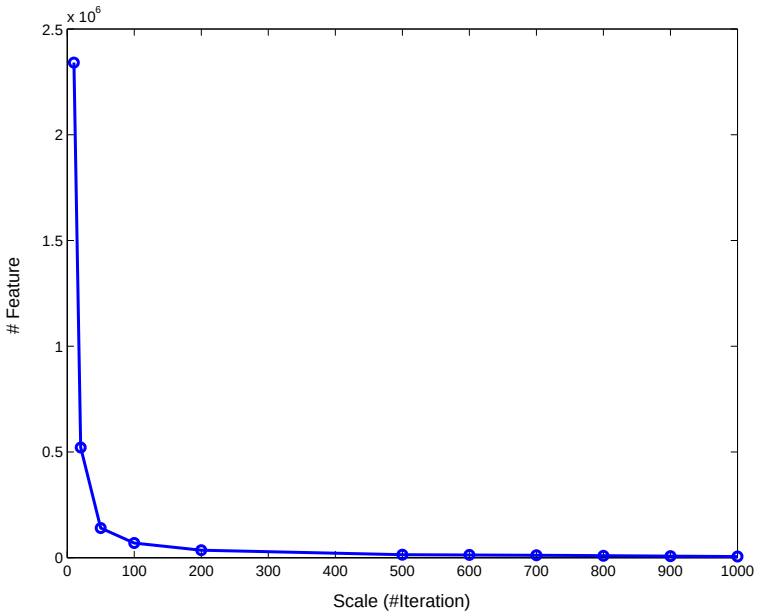
We also attempted to join the whole database with itself using the proposed subseries join approach to cluster motion subseries of different styles. Figures from 6.3 to 6.9 show some results of this subseries join. Motions for running, jumping, and kicking were accurately paired with other similar motions.



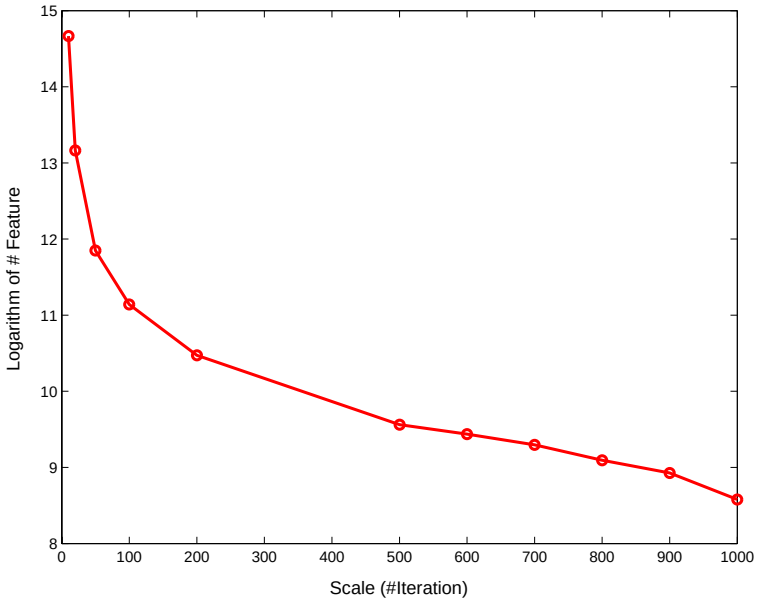
(a) Numbers of features of each channel at different scales.



(b) Natural logarithm of values in (a).



(c) Numbers of features of each channel at different scales.



(d) Natural logarithm of values in (a).

Figure 6.2: Numbers of features at each channel at different scales.

Table 6.1: Performance of the prototype system for a test case involving 100 randomly selected queries, and a whole match of the original database with itself.

Database Parameters	
Number of frames in the database	3,962,581
Number of frames in the query set	20,398
Number of features in the database	101,397
Number of features in the query set	4,879
Computational Time for Matching	
Overall time (min)	3.5
Time per query time series (sec)	2.6
Computational Time for Join	
Overall time (min)	27.8
Time per time series (sec)	0.67

(a) A running motion of 91 frames.

(b) A running motion of 85 frames.

Figure 6.3: Two similar running motions found by the subseries join approach.

(a) A running motion of 91 frames.

(b) A running motion of 109 frames.

Figure 6.4: Two similar running motions found by the subseries join approach.

(a) A jumping motion of 148 frames.

(b) A jumping motion of 126 frames.

Figure 6.5: Two similar jumping motions found by the subseries join approach.

(a) A jumping motion of 143 frames.

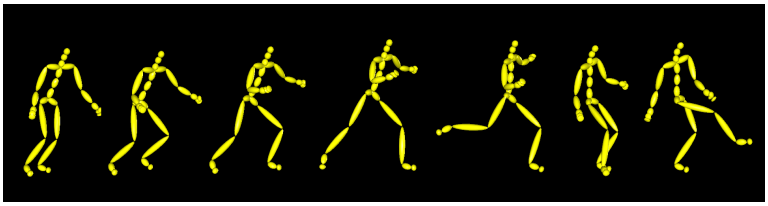
(b) A jumping motion of 126 frames.

Figure 6.6: Two similar jumping motions found by the subseries join approach.

(a) A jumping motion of 148 frames.

(b) A jumping motion of 143 frames.

Figure 6.7: Two similar jumping motions found by the subseries join approach.

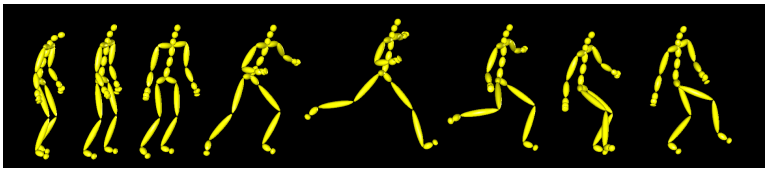


(a) A kicking motion of 119 frames.

(b) A kicking motion of 127 frames.

Figure 6.8: Two similar kicking motions found by the subseries join approach.

(a) A kicking motion of 127 frames.



(b) A kicking motion of 153 frames.

Figure 6.9: Two similar kicking motions found by the subseries join approach.

6.6.2 Compression

Two baseline methods are used for performance comparisons. The first baseline is the cubic interpolating bi-orthogonal wavelet compression (BWC) method [BPvdP07]. BWC is the a reasonable baseline method because it is recent work and has the best results for a temporal coherence scheme to date.

The wavelet coefficient selection in BWC determines how to quantize coefficients optimally. More details can be found in the original paper. The distortion error proposed in [BPvdP07] is used to control the number of iterations. The distortion error metric is defined as

$$\epsilon_x = \sqrt{\frac{1}{n}\sum_{j=1}^{b}\sum_{i=1}^{n}(x_i - x_i')^2\frac{\ell_j}{\ell}}. \tag{6.2}$$

The value x_i is the 3D position of the endpoint of each bone and x_i' is the 3D position of each such endpoint reconstructed from the compressed data. The value b is the number of bones. The value ℓ_j is the length of the bone j, and $\ell = \sum_{j=1}^{p}\ell_j$. In other words, this is a weighted error metric where longer bones are given more weight.

The 3D position values are used represented by the $x-y-z$ coordinate of each joint ends relative to the world space. It is computed from a series of matrix multiplications. The iterations for all DOFs are performed simultaneously since the error metric applies to the whole model. For each joint j, the compression error is computed as follows:

$$\epsilon_j = \frac{1}{n}\sum_{i=1}^{n}(x_i - x_i')^2\frac{\ell_j}{\ell}. \tag{6.3}$$

When $\epsilon_j > w_j E^2/p$, the iterations for joint j are halted. The value E is the upper bound on the reconstruction error, which indicates $\epsilon_x \leq E$. The values w_j are the weights assigned to each joint. The values $0 \leq w_j \leq 1$ and $\sum_{j=1}^{b} w_j = 1$.

Unfortunately, the paper [BPvdP07] did not present many examples suitable for

direct comparison. Therefore, another baseline method, Haar Wavelet compression (HWC), which is similar to BWC, was also used. HWC transforms the original motion data using the simpler Haar wavelet basis but uses the same coefficient selection method as BWC.

The same motion examples as Beaudoin et al. [BPvdP07] were used to compare the proposed compression algorithm (called NSC[1]) with BWC. All compressed results are further coded using *gzip*, which yields an additional 1.1:1 compression ratio. Table 6.2 shows that on average, over the various error rates chosen, NSC achieves 90% and 96% higher compression rates than BWC, for the running motion and the jumping motion, respectively. In other words, NSC nearly doubled the compression rate.

Table 6.2: Compressed size (KB) and compression rates of the baseline method BWC and the proposed method NSC. The uncompressed running motion takes 35.8KB of storage (148 frames). The uncompressed jumping motion takes 505KB of storage (2085 frames).

File	**Method**	**BWC**		**NSC**	
	Error Metric	Size	Rate	Size	Rate
Running	1.40	0.75	48	0.42	85
	0.96	0.94	38	0.52	69
	0.58	1.24	29	0.62	58
	0.26	2.18	16	1.02	35
	0.08	5.08	7.0	2.34	15.3
Jumping	0.67	9.97	51	5.15	98
	0.45	13.5	37	7.54	67
	0.29	18.4	27	9.90	51
	0.14	32.9	15	14.03	36
	0.05	79.2	6.4	32.17	15.7

To broaden the evaluation, the whole database was also compressed. The comparison results are shown in Figure 6.10. NSC is slower than BWC and HWC in compres-

[1]The name "NSC" is derived from *Non-uniform Segmentation Based Compression*.

sion time due to the iterative nature of anisotropic diffusion. However, compression time is not as important as decompression time, which does not involve anisotropic diffusion. The average compression time per frame (using 62 channels) for NSC was about 1 ms. However, the average decompression time for NSC was about 115μs per frame, which is much faster than real time.

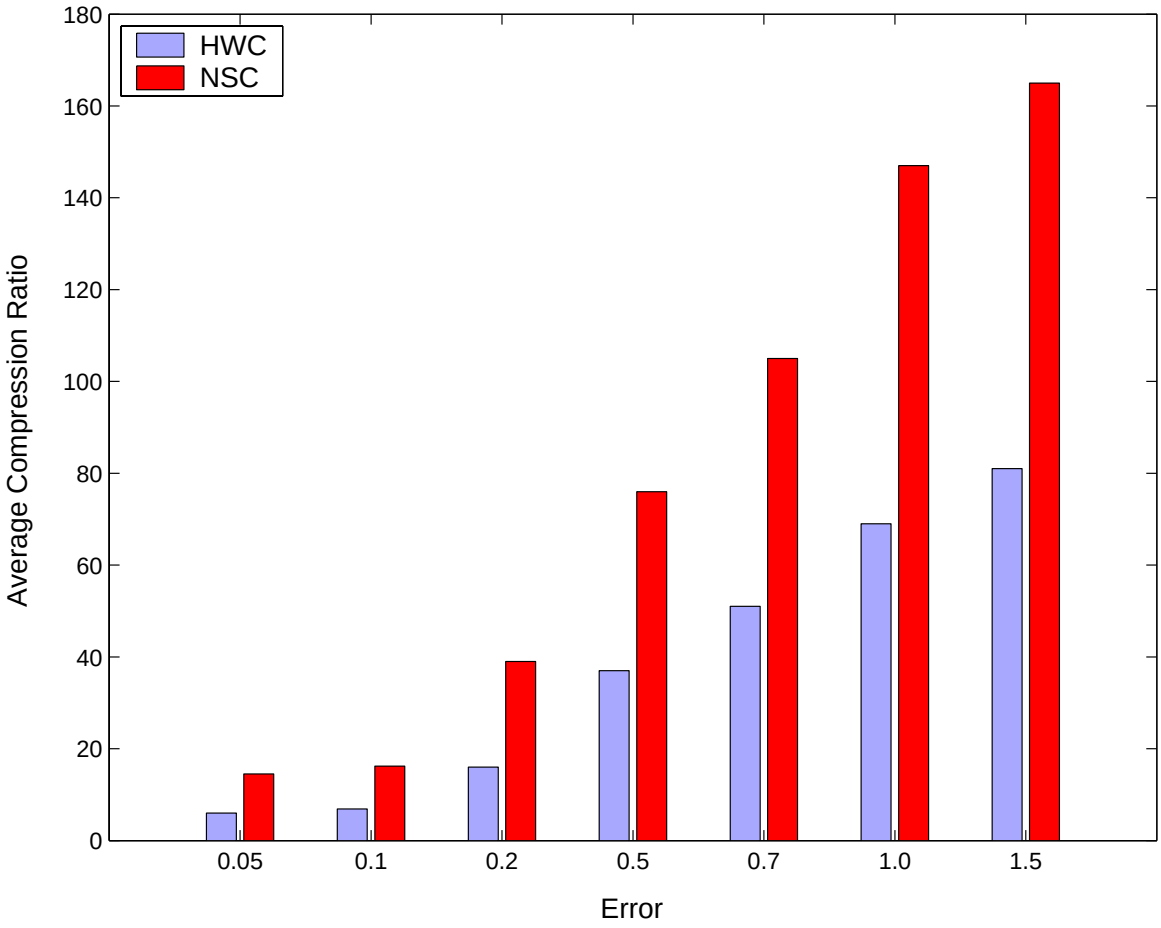


Figure 6.10: The average compression rate of the whole database for each error using baseline method HWC and the proposed method NSC.

6.7 Summary

This chapter experimentally tested the proposed techniques of subseries matching, join, and compression of a large human motion database. The experiments showed the performance results when applying the proposed subseries matching and join approach to a real-world motion capture database. Note that the subseries join approach can also automatically produce clustering results. For example, Figures from 6.3 to 6.5 show a clique where each of the three motions joins the other. The proposed compression approach was compared with the latest related work and an optimized Haar wavelet method. The experiments showed that the proposed approach can achieve about an average 85% higher compression rate than previous work with the same distortion error, and that the compression improvement increases for lower error tolerances. The proposed compression approach is easy to implement and has a fast decompression speed which makes it suitable for game and animation engines.

As mentioned in Section 6.4, the merging method for multi-channel alignment used in my current implementation is not good for multi-channel alignment, although it has found correct results for motion capture data as shown in Section 6.6.1. In my future work, the features should be subdivided according to the discontinuities at all channels so that the discontinuities are alignment along all channels. In this way, each feature is a multi-channel feature. In one feature, the properties of the segments at all channels are taken into account.

Chapter 7

Conclusions and Future Work

This chapter summarizes the main contributions of this thesis and discusses some avenues for future work.

7.1 Summary of Contributions

This thesis investigates using a scale-space analysis to index feature segments of time series datasets for subseries matching and join. The main contributions of this thesis are a new definition of subseries join, which is a generalization of subseries matching and whole matching, and an algorithmic approach to efficiently and accurately solve this problem.

In the proposed techniques, time series data is smoothed and non-uniformly segmented over a scale space by an anisotropic diffusion process. The scale-space analysis generates a hierarchical representation that includes coarse to fine details of the time series. The segments vary in duration but are bounded by significant discontinuities detected by the Canny edge detector. Unlike previous work, the proposed segmentation method is based on the intrinsic structure of time series. Each segment is approximated using a minimal polynomial envelope and other additional parameters, which maps the original data into a reduced-dimensionality space suitable for indexing and

compression. The subseries join approach does not find the exact answer, but it does provide upper and lower bounds of the exact answer.

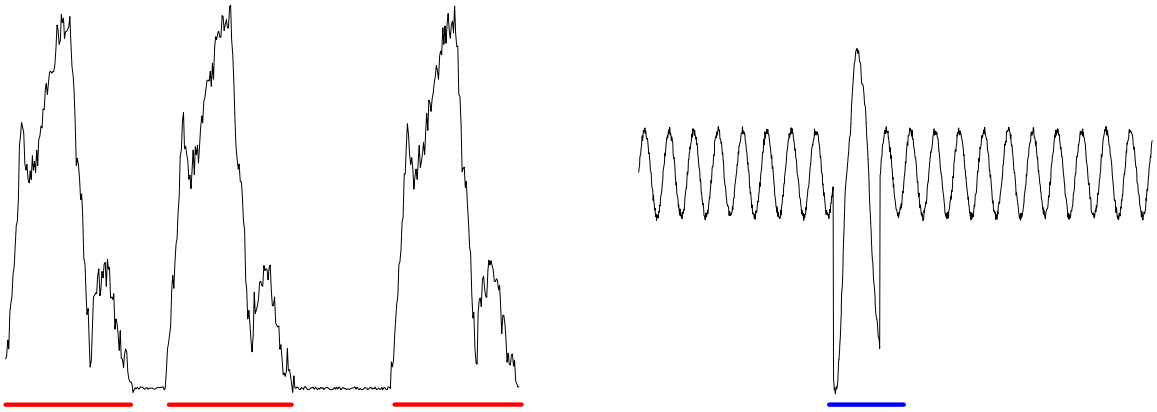
Experiments have demonstrate the effectiveness and efficiency of the proposed subseries matching, join, and compression techniques by testing on both a set of hybrid time series and their variations and a large motion capture database. When applying the proposed techniques to a real-world motion capture database, the prototype system can efficiently search and cluster similar subsequence of motions with a high accuracy rate. Experiments also have demonstrate noise immunity of the proposed techniques compared with previous work. The prototype system can also exploit temporal coherence in the data and achieve a significantly higher compression rate at the same error level than previous work.

7.2 Future Work

The proposed subseries join approach is useful for many data mining applications, including motif detection and anomaly detection. This section introduces the problems of motif detection and anomaly detection and presents some ideas based on the proposed approach to solve these problems.

7.2.1 Motif Detection

Motif detection finds approximately repeated patterns in a time series data. One example is shown in Figure 7.1(a). Yankov et al. proposed a motif detection method that uses a uniform scaling Euclidean distance and a symbolic representation based on thresholding [YKM+07]. The thresholds for converting a time series to a symbolic sequence are heuristically determined. However, the threshold selections may be different with different kinds of data. More importantly, this method is also only semi-automatic, because the user also needs to specify the length of the motif segments manually. Generally, a better definition of a "motif" is needed that does not depend on a priori knowledge of its shape or length.



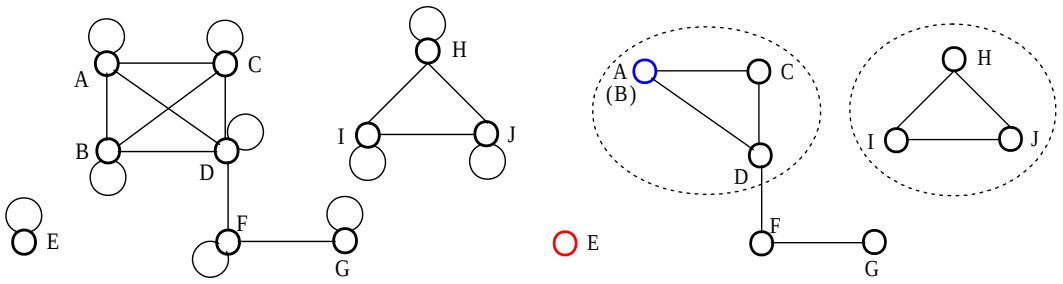
(a) Motif detection. The approximate repeated subseries that are underlined are the motif of this time series.

(b) Anomaly detection. The unusual subseries is underlined.

Figure 7.1: Simulated examples of motif detection and anomaly detection.

Based on the proposed definition of subseries join, we can define motif detection graph-theoretic terms. For example, we can define motif detection as finding k-connected components in the graph given by the subseries join of the time series itself. This is illustrated in Figure 7.2(a). Every vertex represents a subseries. Every edge between two vertices represents a distance between these two subseries that is no greater than a predefined threshold. The proposed approach will always find the subseries itself as one ϵ-similar match. The resulting self-loops from the graph are removed. There may also exist overlapped subseries, for example, A and B. These overlapped subseries are merged by collapsing the associated edges. After this processing, Figure 7.2(a) turns into Figure 7.2(b).

Some alternative graph-theoretic definitions of a motif can also be considered. One possible definition is based on the *maximal cliques* in a graph. A *clique* in an undirected graph is a subgraph in which every vertex is connected to every other vertex in the subgraph. A *maximal clique* is a complete subgraph that is not contained in any other complete subgraph. Unfortunately, the maximal clique problem is one of



(a) The graph of subseries join results.

(b) Remove the self-loops and merges the overlapped vertices A and B.

Figure 7.2: Motif detection and anomaly detection using graph algorithms. The maximal cliques (framed by dashed circles) in the graph of subseries join results give the motifs. The isolated vertex E is an anomaly.

the basic NP-complete problems, and the clique enumeration problem is NP-hard. However, there exist many heuristic algorithms to approximately solve the clique enumeration problem [Akk73, Bys03], including parallel algorithms [DK88, DWX+06] and polynomial-time approximation algorithms [BT00, IIO05].

7.2.2 Anomaly Detection

Anomaly detection finds unusual patterns in a time series data that contains approximately periodic patterns. For example, in Figure 7.1(b), the underlined part is quite different from the other parts of the data that is sine-like.

Based on the proposed definition of subseries join, we can *define anomaly detection as finding the isolated vertices in the graph of subseries join results.* For example, in Figure 7.2, the vertex E is an isolated vertex. Finding isolated vertices is simpler than finding k-connected components in a graph, because it requires only linear computational time.

7.3 Conclusion

This thesis proposed a new definition of subseries join that finds similar subseries in two or more time series datasets, and a solution to compute the subseries join based on a hierarchical feature representation. This thesis also proposed a compression scheme based on the same hierarchical feature representation.

Subseries join is useful for many data mining applications, including clustering, classification, anomaly detection, rule discovery, and motif detection in many domains, such as finance, medicine, music, and motion capture. Chapter 6 has shown some results of using the proposed techniques to cluster motion capture data. This chapter also has discussed using the proposed techniques to solve the problems of motif detection and anomaly detection. In my future work, I will investigate usefulness of the proposed techniques to other applications and other domains.

Bibliography

[Acc] Acclaim. *Acclaim ASF/AMC*. `http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html`.

[ACCO05] J. Assa, Y. Caspi, and D. Cohen-Or. Action synopsis: pose selection and illustration. In *Proceedings of ACM SIGGRAPH 2005*, pages 667–676, 2005.

[AF02] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH 2002*, pages 483–490, 2002.

[AFO03] O. Arikan, D. A. Forsyth, and J. Obrien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3):402–408, 2003.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organizations and Algorithms*, pages 69–84. Springer Verlag, 1993.

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[AJB97] H. Andŕe-Jönsson and D. Z. Badal. Using signature files for querying time-series data. In *Proceedings of 1st European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 211–220, 1997.

[Akk73] E.A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2:1–6, 1973.

[Alt91] S.F. Altschul. Amino acid substitutions matrices from an information theoretic perspective. *Journal of Molecular Biology*, 219:555–665, 1991.

[APWZ95] R. Agrawal, G. Psaila, E. Wimmers, and M. Zaït. Querying shapes of histories. In *Proceedings of 21st International Conference on Very Large Databases*, pages 502–514, 1995.

[Ari06] O. Arikan. Compression of motion capture databases. In *Proceedings of ACM SIGGRAPH 2006*, pages 890–897, 2006.

[BBK01] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[BGL+98] C. H. Bennett, P Gacs, M Li, P Vitanyi, and W Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4):1407–1423, 1998.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[BO99] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.

[BPvdP07] P. Beaudoin, P. Poulin, and M. van de Panne. Adapting wavelet compression to human motion capture clips. In *Proceedings of Graphics Interface 2007*, pages 643–648, 2007.

[BSP+04] J. Barbič, A. Safonova, J. Pan, C. Faloutsos, J. Hodgins, and N. Pollard. Segmenting motion capture data into distinct behaviors. In *Proceedings of Graphics Interface 2004*, pages 185–194, 2004.

[BT00] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 503–515, 2000.

[Bys03] J.M. Byskov. Algorithms for k-colouring and finding maximal independent sets. In *Proceedings of the 14th ACM and SIAM Symposium on Discrete Algorithms*, pages 456–457, 2003.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[CF99] K. Chan and A. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, pages 126–133, 1999.

[CH05] J. Chai and J. Hodgins. Performance animation from low-dimensional control signals. In *Proceedings of ACM SIGGRAPH 2005*, pages 686–696, 2005.

[Chu92] C. K. Chui. *An Introduction to Wavelets*. Academic Press, San Diego, CA, 1992.

[CL94] W. Chang and E. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.

[CMTV00] A. Corral, Y. Manolopoulos, Y. Theodorisdis, and M. Vassilakopoulos. Closest pair queries in spatial databases. In *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data*, pages 189–200, 2000.

[CPB+98] E. Caiani, A Porta, G. Baselli, M. Turiel, S. Muzzupappa, F. Pieruzzi, C. Crema, A Malliani, and S. Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. *IEEE Computers in Cardiology*, 25:73–76, 1998.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Databases*, pages 426–435, 1997.

[CV05] Rudi Cilibrasi and Paul M.B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[CVB+03] M. Cardle, M. Vlachos, S. Brooks, E. Keogh, and D. Gunopulos. Fast motion capture matching with replicated motion editing. In *Proceedings of ACM SIGGRAPH 2003 Technical Sketches and Applications*, 2003.

[DK88] E. Dahlhaus and M. Karpinski. A fast parallel algorithm for computing all maximal cliques in a graph and the related problems. In *No. 318 on SWAT 88: The 1st Scandinavian Workshop on Algorithm Theory*, pages 139–144, London, UK, 1988. Springer-Verlag.

[DSO78] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5(3):345–352, 1978.

[DWX+06] N. Du, B. Wu, L. Xu, B. Wang, and X. Pei. A parallel algorithm for enumerating all maximal cliques in complex network. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 320–324, 2006.

[Elk03] C. Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pages 147–153, 2003.

[Far01] G.E. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann, 2001.

[FF05] K. Forbes and E. Fiume. An efficient search algorithm for motion data using weighted PCA. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–76, 2005.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data*, pages 419–429, 1994.

[FTJF01] R. Filho, A. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: the omni family of all-purpose access methods. In *Proceedings of 17th International Conference on Data Engineering*, pages 623–630, 2001.

[GBT04] P. Glardon, R. Boulic, and D. Thalmann. A coherent locomotion engine extrapolating beyond experimental data. In *Computer Animation and Social Agents*, pages 73–84, 2004.

[GCB92] G.H. Gonnet, M.A. Cohen, and S.A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[GD95] D. Gavrila and L. Davis. Towards 3D model-based tracking and recognition of human movement: a multi-view approach. In *Proceedings of the IEEE Workshop on Face and Gesture Recognition*, pages 272–277, 1995.

[GK04] I. Guskov and A. Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 183–192, 2004.

[GMHP04] K. Grochow, S. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. In *Proceedings of ACM SIGGRAPH 2004*, pages 522–531, 2004.

[GP90] Z. Galil and K. Park. An improved algorithm for approximate string matching. *SIAM Journal of Computing*, 19(6):989–999, 1990.

[GP95] K. Gollmer and C. Posten. Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. In A. Morris and E. Martin, editors, *On-Line Fault Detection and Supervision in the Chemical Process Industries*. ACM Press, New York, NY, 1995.

[Gra] Graphics Lab, Carnegie-Mellon University. *Carnegie-Mellon MoCap Database*. http://mocap.cs.cmu.edu.

[GSK87] S. Gupta, K. Sengupta, and A. A. Kassim. Compression of dynamic 3D geometry data using iterative closest point algorithm. *Computer Vision and Image Understanding*, 87(1-3):116–130, 1987.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[HH92] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *The Proceedings of the National Academy of Sciences Online (US)*, 89(biochemistry):10915–10919, 1992.

[HKK07] T. S. Han, S.-K. Ko, and J. Kang. Efficient subsequence matching using the longest common subsequence with a dual match index. In *Machine Learning and Data Mining in Pattern Recognition*, volume 4571/2007 of *Lecture Notes in Computer Science*, pages 585–600. Springer Berlin/Heidelberg, 2007.

[HLB94] A. Hanjalic, R. Lagendijk, and J. Biemond. Improving text retrieval for routing problem using latent semantic indexing. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 282–291, 1994.

[HY99] Y. Huang and P. Yu. Adaptive query processing for time-series data. In *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 282–286, 1999.

[IIO05] H. Ito, K. Iwama, and T. Osumi. Linear-time enumeration of isolated cliques. In *Proceedings of 13rd Annual European Symposium on ALgorithms (ESA '05)*, volume 3669/2005 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2005.

[IR03] L. Ibarria and J. Rossignac. Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 126–135, 2003.

[KAS98] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 166–176, 1998.

[KCMP01] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 151–162, 2001.

[Keo02] E. J. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th Conference on Very Large Databases*, pages 406–417, 2002.

[Keo03] E. J. Keogh. Efficiently finding arbitrarily scaled patterns in massive time series databases. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 253–265, 2003.

[Keo06a] E. Keogh. *The UCR Time Series Data Mining Archive*. Department of Computer Science and Engineering, University of California, Riverside, 2006. `http://www.cs.ucr.edu/~eamonn/TSDMA/`.

[Keo06b] E. J. Keogh. Time series tutorial. `http://www.cs.ucr.edu/~eamonn/tutorials.html`, 2006.

[KG04] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *Proceedings of ACM SIGGRAPH 2004*, pages 559–568, 2004.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH 2002*, pages 473–482, 2002.

[KJF97] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 289–300, 1997.

[KM04] K. Kondo and K. Matsuda. Keyframes extraction method for motion capture data. *Journal for Geometry and Graphics*, 8(1):81–90, 2004.

[KP00] E. J. Keogh and M. Pazzani. Scaling up dynamic time warping for data mining applications. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 285–289, 2000.

[KPC01] S. W. Kim, S. Park, and W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 607–614, 2001.

[KPS03] T.-H. Kim, S. I. Park, and S. Y. Shin. Rhythmic-motion synthesis based on motion-beat analysis. In *Proceedings of ACM SIGGRAPH 2003*, pages 392–401, 2003.

[KS97] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 24–30, 1997.

[LCL+04] M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi. The similarity metric. *IEEE Transaction Information Thoery*, 50:3250–3264, 2004.

[LCR+02] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH 2002*, pages 491–500, 2002.

[Len99] J. E. Lengyel. Compression of time-dependent geometry. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 89–95, 1999.

[LGC94] Z. Liu, S. Gortler, and M. Cohen. Hierarchical spacetime control. In *Proceedings of ACM SIGGRAPH 1994*, pages 35–42, 1994.

[Li07] M. Li. Information distance and its applications. *International Journal of Foundations of Computer Science*, 4094:1–9, 2007.

[Lin06] Y. Lin. Efficient motion search in large motion capture databases. In *Proceedings of the 2nd International Symposium on Visual Computing (ISVC '06)*, pages 151–160. LNCS 4291, 2006.

[LKLP02] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proceedings of the 2nd International Workshop Temporal Data Mining*, pages 370–377, 2002.

[LM06] G. Liu and L. McMillan. Segment-based human motion compression. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 127–135, 2006.

[LM07] Y. Lin and M. D. McCool. Nonuniform segment-based compression of motion capture data. In *Proceedings of the 3rd International Symposium on Visual Computing (ISVC '0y)*, pages 56–65, 2007.

[LS99] J. Lee and S. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 1999*, pages 39–48, 1999.

[LS00] J. Lee and S. Shin. Multiresolution motion analysis with applications. In *International Workshop on Human Modeling and Animation*, pages 131–143, 2000.

[LV97] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 2nd edition, 1997.

[LWS02] Y. Li, T. Wang, and H. Y. Shum. Motion texture: a two level statistical model for character motion synthesis. In *Proceedings of ACM SIGGRAPH 2002*, pages 465–472, 2002.

[LZWP03] F. Liu, Y. Zhuan, F. Wu, and Y. Pan. 3D motion retrieval with motion index tree. *Computer Vision and Image Understanding*, 92(2-3):265–284, 2003.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. In *Proceedings of the Royal Society*, pages 287–217, 1980.

[MP79] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 204(1156):301–328, 1979.

[MR81] C. Myers and L. Rabiner. A level building programming dynamic time warping algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29(2):284–297, 1981.

[MRG05] M. Müller, T. Röder, and M. Glausen. Efficient content-based retrieval of motion capture data. In *Proceedings of ACM SIGGRAPH 2005*, pages 677–685, 2005.

[MRK+07] A. McGovern, D. Rosendahl, A. Kruger, M. Beaton, R. Brown, and K. Droegemeier. Understanding the formation of tornadoes through data mining. In *Proceedings of the 5th Conference on Artificial Intelligence and its Applications to Environmental Sciences at the American Meteorological Society*, 2007.

[MWH02] Y-S. Moon, K-Y. Whang, and W.-S. Han. General match: a subsequence matching method in time-series databases based on generalized windows. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 382–393, 2002.

[MWL01] Y-S. Moon, K-Y. Whang, and W-K Loh. Duality-based subsequence matching in time-series databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 263–272, 2001.

[MYAU01] Y. Morinaka, M. Yoshikawa, T. Amagasa, and S. Uemura. The lindex: an indexing structure for efficient subsequence matching in time sequence databases. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 51–60, 2001.

[Nav02] G. Navarro. Searching in metric spaces by spatial approximation. In *Proceedings of the 23rd Conference on Very Large Databases*, pages 426–435, 2002.

[NW70] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[OS75] A. Oppenheim and R. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1975.

[Pav73] T. Pavlidis. Waveform segmentation through functional approximation. *IEEE Transactions on Computers*, C-22(7):689–697, 1973.

[PB02] K. Pullen and C. Bregler. Motion capture assisted animation: texturing and synthesis. In *Proceedings of ACM SIGGRAPH 2002*, pages 501–508, 2002.

[PC03] S. Park and W. Chu. Similarity-based subsequence search in image sequence databases. *International Journal of of Images and Graphics*, 3(1):31–53, 2003.

[PD96] J. M. Patel and D. J. Dewitt. Partition based spatial-merge join. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 259–270, 1996.

[PM90] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[PM01] I. Popivanov and R. Miller. Similarity search over time series data using wavelets. In *Proceedings of the 17th International Conference on Data Engineering*, pages 212–221, 2001.

[PRM00] V. Pavlović, J. M. Rehg, and J. MacCormick. Learning switching linear models of human motion. In *Proceedings of Neural Information Processing Systems (NIPS)*, pages 981–987, 2000.

[QWW98] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proceedings of the 7th International Conference on Information and Knowledge Management*, pages 251–258, 1998.

[Rap] RapidMind Inc. *RapidMind Multi-Core Development Platform*. `http://www.rapidmind.com`.

[RCB98] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–41, 1998.

[RJ93] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[SBNW96] S. Smoliar, J. Baker, T. Nakayama, and L. Wilcox. Multimedia search: an authoring perspective. In *Proceedings of the 1st International Workshop on Image Databases and Multimedia Search (IAPR)*, pages 1–8, 1996.

[SC78] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[SC03] S. Shekar and S. Chawla. *Spatial Databases: a Tour.* Prentice Hall, 1st edition, 2003.

[Sel74] P. Sellers. On the theory and computation of evolutionary distances. *SIAM Journal of Applied Mathematics*, 26:787–793, 1974.

[Sel80] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Algorithmica*, 1:359–373, 1980.

[SHP04] A. Safonova, J. Hodgins, and N. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. In *Proceedings of ACM SIGGRAPH 2004*, pages 514–521, 2004.

[SOC99] M. Schmill, T. Oates, and P. Cohen. Learned models for continuous planning. In *Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics*, pages 278–282, 1999.

[SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[SZ96] H. Shatkay and S.B. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the 12nd International Conference on Data Engineering*, pages 536–545, 1996.

[TB99] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society*, 61(3):611–622, 1999.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the 6th International Conference on Computer Vision*, pages 836–846, 1998.

[Uhl91] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letter*, 40(4):175–179, 1991.

[Ukk85] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[VHGK03] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *Proseedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 216–225, 2003.

[WAA00] Y. Wu, D. Agrawal, and A. Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, pages 488–495, 2000.

[WB03] J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2003*, pages 232–238, 2003.

[Wit83] A. Witkin. Scale-space filtering. *Proceedings of the International Joint Conference on Artificial Intelligence*, 2:1019–1022, 1983.

[WKX06] L. Wei, E. J. Keogh, and X. Xi. SAXually explicit images: finding unusual shapes. In *Proceedings of International Conference on Data Mining (ICDM '06)*, pages 711–720, 2006.

[WW03] T. S. F. Wong and M. H. Wong. Efficient subsequence matching for sequences databases under time warping. In *Proceedings of Database Engineering and Applications Symposium(IDEAS'03)*, pages 139–148, 2003.

[XKS+06] X. Xi, E. J. Keogh, C. R. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pages 1033–1040, 2006.

[YF00] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary L_p norms. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 385–394, 2000.

[YJF98] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*, pages 201–208, 1998.

[YKM+07] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13rd International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD '07)*, 2007.

[YKM+08] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. The asymmetric approximate anytime join: a new primitive with applications to data mining. In *Proceedings of International Conference on Data Mining (SDM '08)*, 2008.

[ZHZL07] X. Zhang, Y. Hao, X. Zhu, and M. Li. Information distance from a question to an answer. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 874–883, 2007.

[ZMCF05] V.B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. In *Proceedings of ACM SIGGRAPH 2005*, pages 697–701, 2005.