The Use of Bioinformatics Techniques to Perform Time-Series Trend Matching and Prediction

Mark Transell

The Use of Bioinformatics Techniques to Perform Time-Series Trend Matching and Prediction

by

Mark Transell

A dissertation submitted in partial fulfillment
of the requirements for the degree

Master of Engineering (Control Engineering)

in the

Department of Chemical Engineering
Faculty of Engineering, the Built Environment and Information Technology

University of Pretoria
Pretoria

December 2012

SYNOPSIS

Process operators often have process faults and alarms due to recurring failures on process equipment. It is also the case that some processes do not have enough input information or process models to use conventional modelling or machine learning techniques for early fault detection.

A proof of concept for online streaming prediction software based on matching process behaviour to historical motifs has been developed, making use of the Basic Local Alignment Search Tool (BLAST) used in the Bioinformatics field. Execution times of as low as 1 second have been recorded, demonstrating that online matching is feasible.

Three techniques have been tested and compared in terms of their computational efficiency, robustness and selectivity, with results shown in Table 1:

- Symbolic Aggregate Approximation combined with PSI-BLAST
- Naive Triangular Representation with PSI-BLAST
- Dynamic Time Warping

Table 1: Properties of different motif-matching methods

Property	SAX-PSIBLAST	TER-PSIBLAST	DTW
Noise tolerance (Selectivity)	Acceptable	Inconclusive	Good
Vertical Shift tolerance	None	Perfect	Poor
Matching speed	Acceptable	Acceptable	Fast
Match speed scaling	$O < O(mn)$	$O < O(mn)$	$O(mn)$
Dimensionality Reduction Tolerance	Good	Inconclusive	Acceptable

It is recommended that a method using a weighted confidence measure for each technique be investigated for the prupose of online process event handling and operator alerts.
Keywords: SAX, BLAST, motif-matching, Dynamic Time Warping

ACKNOWLEDGEMENTS

I would like to acknowledge the endless patience and support of my supervisor Carl Sandrock, the support of my family and friends, and the terrible strain placed upon the Oxford comma in this sentence. A further thanks to Prof. Eamonn Keogh for freely sharing testing data online, and to Sasol for providing the funding which made this project possible.

I was not trying to predict the future. I was trying to prevent it.

Ray Bradbury

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

α	Autoregression parameter
β_i	SAX breakpoint (time-series value)
ϵ	White-noise process for AR model
γ	Smoothing factor for low-pass filter
∂	Partial derivative operator
σ	Standard deviation (Normally distributed)
A	Character sequence
B	Character sequence
C	The length of a DTW warping path
$cost_{i,j}$	Scoring matrix value at point i, j
gap_c	Gap cost in SWALE algorithm
H	Smith-Waterman score matrix
$H_{i,j}$	Smith-Waterman score matrix element i, j
k	Length of deletions in character sequence
l_{max}	Maximum length of stored motif data vectors
M	Size of boxcar filter window
m	Time series length

n	Data vector length / Character sequence length
O	Big O notation: processing time
P	State transition probability matrix for HMM
p	Autoregressioon model order
p_s	Current time value for CQP
$P_{i,j}$	Probability of state transition from state X_i to state X_j
Q	DTW warping path
q_c	The c^{th} element of a DTW warping path
$reward_m$	Match reward value in SWALE
S	Cumulative match score: BLAST
T	Threshold value for BLAST score
t	Time
U	Similarity score in scoring matrix
w_i	Value of weighting function at point i
W_k	Smith-Waterman weighting for k-length deletion
X	Maximum allowable score decrease for BLAST gap extension
X	Process State
x	Time series data vector
x_i	Time series data value at point $i > 0$
y	Time series data vector
y_i	Time series data value at point $i > 0$
Z	DTW dynamic programming matrix
BP	Abbreviation for Breakpoints
DB	Abbreviation for Database Length
QL	Abbreviation for Query Length

Res1	No dimensionality reduction applied
Res20	20-Fold dimensionality reduction applied
SAX10	SAX performed with 10 breakpoints
SAX20	SAX performed with 20 breakpoints

CHAPTER 1

INTRODUCTION

Process engineers and operators have more access to historical plant data than ever before. It is important for operators to be alerted when undesirable plant behaviour is occuring, so that they can take action to prevent process faults. In an industrial setting, it is occasionally the case that certain process inputs are not known, or are not measured, and no dynamic model of the process exists. In such a case, any predictions would have to be based on historical data for process outputs, rather than model-based prediction methods. Although automated techniques already exist to cluster and classify process events (Labuschagne, 2008), time-series motif matching may give additional insight into recurring undesirable behaviour. Predictions for future process behaviour could improve the ability of operators to take corrective action before a process reaches an undesirable state.

Dynamic Time Warping (DTW) was first applied to speech-recognition (Sakoe & Chiba, 1978), but has since been applied to many fields, including time-series motif-matching (Rath & Manmatha, 2003; Santosh, 2010).

Sequence matching is also an important part of bioinformatics; a field which has received a marked increase in research funding and attention in recent times (Molatudi et al., 2009), helped in part by the publicity received by the Human Genome Project. This has led to the development of highly efficient algorithms and automated software implementations.

These freely avalable algorithms promise easier implementation, benefits to signal noise tolerance and improved gap handling to DTW and similar methods for motif-matching. Time series need to be converted into character strings before bioinformatics techniques can be applied to them. In this work, Symbolic Aggregate Approximation (SAX) and a naive version of Triangular Episodic Representation (TER) are used to convert the time series into strings, and PSI-BLAST is used to match these sequences.

These SAX-PSIBLAST and TER-PSIBLAST techniques are compared to DTW in terms of computational expense, selectivity and matching power. Based on these comparison results, a method is proposed whereby predictions of future process behaviour can be generated based on matches to historical data without explicitly requiring input information. A proof of concept is provided for this online motif-matching procedure. This is intended to form part of a fault detection system for plant operators, where process predictions can be used as a criterion for alerts.

Part I

Theory

CHAPTER 2

TIME-SERIES AND MOTIFS

2.1 Time-Series Data Representation

Time-series prediction techniques have been used extensively in the fields of economics, engineering, computer science, meteorology, medicine and bioinformatics. Time series data can include electrocardiograms (ECG), daily temperature, stock prices (Fu, 2011) and chemical process data.

Time series data can be represented in computational space as a discrete vector of x values with corresponding t values in another discrete vector. A time series of length n can be represented by a vector $[x_0, x_1, \ldots x_n]$. There is usually a constant sampling rate associated with any given time series, which means that all consecutive data points are equidistant on the time axis. This is not always the case in practice, but when it occurs that data points are not evenly spaced, interpolation is often used to resample the data. It will be assumed for the purposes of this study that all time-series used have been interpolated or resampled to have linearly-spaced time vectors associated with the time series. This is a requirement for most distance measures to have meaning when applied to said time series (Gao & Wang, 2002).

It is possible to use discrete models to predict future process behaviour based on process input information. Indeed, this is how most Model-Predictive Controllers (MPCs) operate. A problem arises when the exact nature of the inputs are not known, or there are certain abnormal patterns in process behaviour that occur due to a fault that is not picked up by any sensor attached to a known process input. In these cases, databases of process motifs must be mined continuously for predictions about future process behaviour.

2.2 Process Motifs

Before motif-matching can be used, it is necessary to first *define* what is meant by a motif. Chiu et al. (2003) mentions several definitions of a motif. The most applicable to this work are:

Prototypes Several time series classification algorithms work by constructing typical *prototypes* of a class of behaviour. These prototypes may be considered motifs (Keogh & Pazzani, 1998).

Typical shapes Many time series anomaly/interestingness detection algorithms essentially consist of modeling normal behavior with a set of typical shapes (seen here as *motifs*), and detecting patterns which are dissimilar to these motifs (Dasgupta & Forrest, 1995).

Approximate periodic patterns Much work on finding approximate periodic patterns in time series can viewed as an attempt to discover motifs that occur at constrained intervals (Han et al., 1999).

For the purpose of online matching and prediction, it may be considered that even trivial shapes (straight lines) can be considered motifs, since process prediction would be desired even for "uninteresting" shapes. This implies that although in most cases the motifs searched for would fall into one of the three categories defined above, any subsequence of a time-series will be considered a motif for predictive purposes.

A good introductory example into this kind of continuous motif-matching can be found in the Continous Query Process (CQP), which was implemented by (Gao & Wang, 2002).

2.3 Continuous Query Process (CQP) (Gao & Wang, 2002)

Gao and Wang (2002) were able to develop a continous query process whereby a streaming dataset could be buffered, and the buffered data could be checked against a database of indexed motifs of process behaviour. The CQP process checks for cross-correlation between the Fast Fourier Transforms (FFTs) of the stored motifs and of the buffered data, in order to sift out candidates which can be used to flag abnormal situations.

Figure 2.1 demonstrates what is expected by the CQP process, given a particular process prediction. In the figure, p_s refers to the current time, and l_{max} refers to the maximum length of all stored motifs. and process output values between p_s and $p_s + n$ are predicted by using an n-step ahead prediction model. The range of real values from

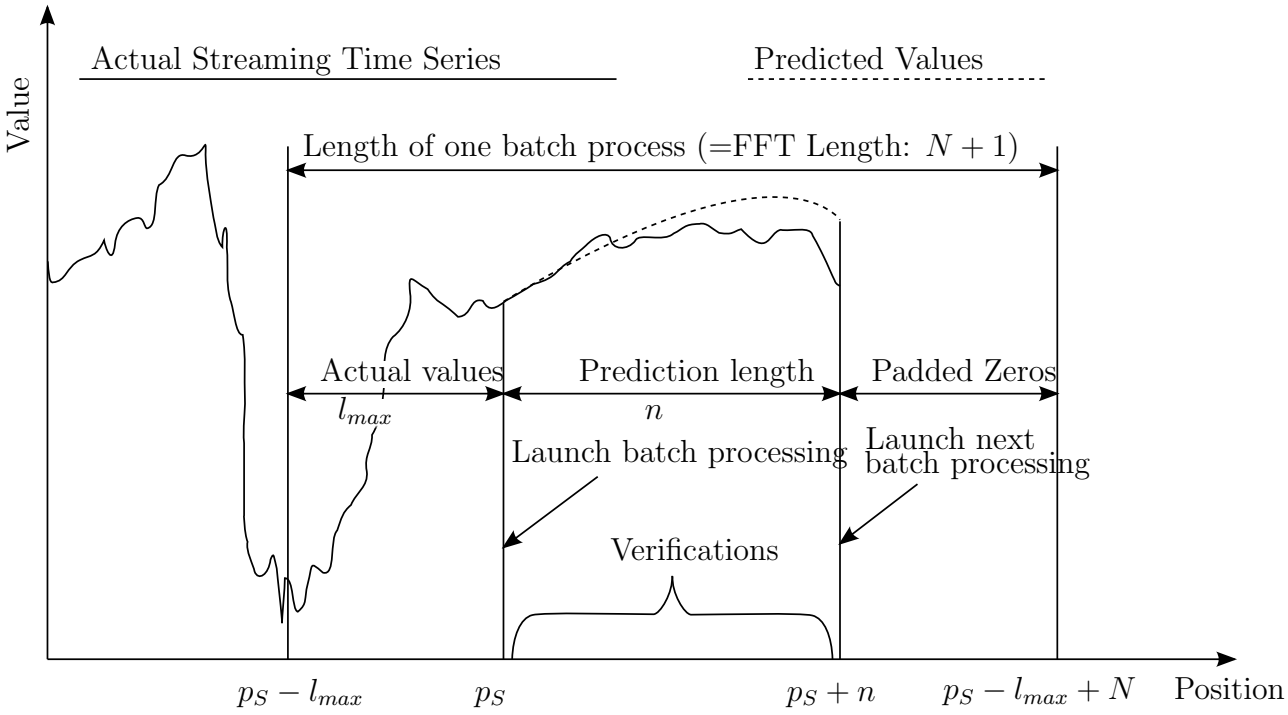

Figure 2.1: An illustration of streaming time-series prediction based on motif-matching techniques (Gao & Wang, 2002).

$p_s - l_{max}$ to $p_s - 1$ is concatenated with the predicted values from p_s to $p_s + n$, and used for matching with stored motifs.

The inclusion of the predicted values in the motifs to be matched allows for Fast Fourier Transform (FFT) batch techniques to be used to calculate *predicted distances* between the streaming data series and the pattern series matched. This allows for the prediction error to be used to obtain the neighbors of the streaming series for matching. However, this method is unsuitable for cases where no process model exists. This problem can be resolved by using only the actual time-series values instead of the predicted values, at the cost of computational efficiency.

This is an interesting application of these techniques, but the focus of this dissertation is to answer the question: Could matched motifs themselves be used for process prediction?

Before this question can be answered, it is necessary to examine how process data is usually filtered before storage.

2.4 Time-series Filtering

When mining process data for trends, it is often necessary to reduce measurement and process noise by using filters. There are numerous filtering methods available, each applicable in certain situations. Due to time constraints, only two filtering methods are examined in this study.

2.4.1 Boxcar Filtering

The boxcar filter, or moving average filter is the most common filter in digital signal processing, mainly because it is the easiest digital filter to understand and use. The moving average filter is optimal for reducing random noise while retaining a sharp step response (Smith, 1999).

The boxcar filter operates by the application of Equation 2.1 (Smith, 1999). This is the one-sided boxcar filter, but on a stored dataset a two-sided filter can be applied, as in Equation 2.2

$$y_i = \frac{1}{M} \sum_{j=0}^{M-1} x_{i+j} \tag{2.1}$$

$$y_i = \frac{1}{M} \sum_{j=1}^{M/2} (x_{i+j} + x_{i-j}) + x_i/M \tag{2.2}$$

However, the boxcar filter is a poor filter for frequency domain encoded signals, with little ability to separate one band of frequencies from another. Filters such as the Gaussian, and multiple-pass moving average have slightly better performance in the frequency domain, at the expense of increased computation time due to convolution (Smith, 1999).

2.4.2 Low-Pass Filtering

The low-pass filter method is a filtering method by which high frequencies normally associated with process noise are attenuated. In discrete time series, this is also referred to as the exponential moving average (Brown, 1956). Equation 2.3 shows how this filter is calculated, where γ is the smoothing factor, related to the cutoff frequency for the filter. The original dataset is represented as x and the filtered dataset is y.

$$y_t = \gamma x_{t-1} + (1 - \gamma) y_{t-1} \tag{2.3}$$

However, due to the computational expense of convolution, in most cases it is faster to use recursive boxcar filtering to reduce noise (National Institute of Standards and Technology, 2012). For this reason, boxcar filtering is the preferred technique used in this dissertation.

CHAPTER 3

CONVENTIONAL MODELLING AND PREDICTION TECHNIQUES

3.1 Computational Complexity: Big O Notation

Before discussing more complex tasks than time-series filtering, some terminology regarding computational complexity and speed is necessary. When describing the speed of an algorithm, it is useful to think of the time taken for the algorithm to complete in the worst case situation. Definition 1 describes a bounding envelope for this completion time:

Definition 1 *Given a function f(N), O(f(N)) denotes the set of all g(N) such that* $|g(N)/f(N)|$ *is bounded from above as* $N \to \infty$ *(Sedgewick & Flajolet, 1996).*

This big O notation provides a way to express an upper bound for processing time. For complexity studies, the importance of this notation is that it allows implementation details to be hidden by ignoring constant factors (Sedgewick & Flajolet, 1996). If a function describing the processing time of an algorithm is described as $f(N)$ where N is some algorithmic input, and $f(N)$ expresses an asymptotic bound of $k \log N$ with k being some constant, the computational complexity of the algorithm can be notated as $O(\log N)$.

It must be noted that big O notation refers only to the behaviour of the bounding function; the algorithm might complete sooner than the bounding time, but not after. It must also be noted that this behaviour is *asymptotic*, meaning that the fastest-growing contribution to processing time will dominate as $N \to \infty$. The computational complexity of all algorithms examined in this disertation are described using the big O notation.

3.2 Neural Networks (Hopfield, 1982)

Neural Networks are systems which were developed from their inception (McCulloch & Pitts, 1988) to emulate the processes present in the human brain. The brain is a highly complex, nonlinear and parallel computer, with the ability to organise its structural constituents known as *neurons* to perform certain computational tasks (Haykin, 1999).

Neural networks require training sets in order to build models for a process. Once trained, a neural network could make predictions for future time-series behaviour, based on previous datasets.

However, the learning process is not guaranteed to work in all cases, particularly when no input information is available. Neural Networks also require a backlog of historical data, and may be computationally intensive for complex systems. For these reasons, neural networks will not be discussed further in this study.

3.3 Autoregressive models

In most cases, building a process model to predict future process behaviour requires some data about the inputs to the process. If these data are not available, autoregressive models are used to forecast future process outputs.

Autoregressive models can be derived for a subset of processes known as *stationary stochastic* processes. A stationary process is one for which any two subsets of the output time-series have the same probability distribution if they are of the same length, i.e. the probability distribution for the outputs does not change over time (Pollock, 1999). Vector autoregression (VAR) constructs a model of a stationary process using the assumption that all inputs to the process are *normally* distributed.

The p^{th}-order autoregressive process, or AR(p) process $x(t)$, is defined by Equation 3.1, where α_i is the i^{th} autoregressive model parameter. The value for ϵ represents the white-noise input of the process, with an expected value of 0 (Pollock, 1999).

$$\alpha_0 x_t + \alpha_1 x_{t-1} + \cdots + \alpha_p x_{t-p} = \epsilon_t. \tag{3.1}$$

The AR method is not a motif-matching method, but instead builds a process model from historical data to predict future behaviour. This means that AR models are *data-dependent*, and are not applicable to every process. However, this method has some limited application in process control, since a continuously controlled process without set-point changes and with natural disturbances that follow a Gaussian distribution is an example of a wide sense stationary stochastic process (Brockwell & Davis, 1991). If a multivariable process is stationary and if the statistical dependencies between widely separated elements are weak, then it is possible to estimate consistently a limited number

of autocovariances which express the statistical dependence of proximate elements of the sequence (Pollock, 1999). For this reason, it is considered here that AR models are more applicable for estimating state dependencies than for predicting the behaviour of a single process output.

Autoregression is often combined with a moving average (MA) model (Pollock, 1999; Brockwell & Davis, 1991), which makes use of input data to develop the process model. It is often the case that the inputs to a chemical process are unknown. Therefore, the inclusion of MA modelling makes for unwelcome restrictions on the generality of any process-prediction application. For this reason, the ARMA process is excluded from this investigation.

The AR technique is applied, but due to the inapplicability of a MA model, it will have to be of a high order to make acceptable predictions for arbitrary process behaviour (Pollock, 1999).

3.4 Hidden Markov Models

Another method that can be used to model processes with unknown inputs is Markov modelling. Markov models are based on the assumption that for any given state of a system X_i, there is a definite probablity $P_{i,j}$ that in the next time interval the system state will change to X_j (Ross, 1996). This leads to a one-step transition probability matrix P, given in Equation 3.2 (Ross, 1996).

$$P = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & \dots & P_{0,j} & \dots \\ P_{1,0} & P_{1,1} & P_{1,2} & \dots & P_{1,j} & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \\ P_{i,0} & P_{i,1} & P_{i,2} & \dots & P_{i,j} & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots \end{bmatrix} \tag{3.2}$$

Markov models can generate probabilistic predictions for future process behaviour, but these predictions are dependent on the accuracy of the transition model for the process. Another problem is that of *hidden* states that are not accessible to direct observation. In chemical processes, there are often several states which are not observed directly, and must be inferred from measured process outputs. Models for these systems are known as *hidden* Markov models (HMM). The Baum-Welch algorithm (Baum et al., 1970) can be used for deriving the transition probabilities and structure for a HMM from training data (Zhang, 2004). Once these transition probabilities have been derived, the Viterbi algorithm (Viterbi, 1967) is a popular method used to infer the HMM sequence which results in an observed sequence of outputs (Zhang, 2004).

These HMM methods are popular in the bioinformatics community, particularly

for determining genetic sequence substitution probabilities and subsequence matching (Zhang, 2004; Durbin et al., 1998).

CHAPTER 4

BIOINFORMATICS TECHNIQUES

Before discussing the sequence-matching algorithms used in the field of bioinformatics, it is necessary to examine the nature of the problems these techniques were developed to solve. The matching of sections of genetic data to databases of such data is of particular interest to this study. The acquainted reader may want to skip Section 4.1, as it is basic knowledge to those in the field.

4.1 Genetic Sequences and String Representation

In the field of bioinformatics, a common task is that of matching genetic material obtained from experimental results to a database of previously documented genetic material. The traditional way of representing the genetic information contained within a DNA (deoxyribonucleic acid) or RNA (ribonucleic acid) molecule is as a character string. This string may only consist of characters representing nucleotide base pairs: 'ACGTU' refers to the nucleotides adenine, cytosine, guanine, thymine and uracil respectively. Uracil is the RNA equivalent of thymine, which is contained only in DNA.

In biological applications, it is known that any particular group of three nucleotides in a row codes for one of twenty amino acids (the building block for proteins) or a 'stop' signal (Gardner, 1975). Each amino acid that is coded for has been allocated a particular symbol. It is not important to know which amino acid is coded for by which letter or which string of three nucleotides codes for which amino acid, only that the list of available characters for us to use is: ACDEFGHIKLMNPQRSTVWY. Using only these characters in strings to describe process behaviour will allow bioinformatics programs to be used without causing errors due to unknown characters. Numerous techniques have been developed in bioinformatics to match efficiently two character strings, or sequences, in order to perform database searches on genetic libraries. This category of sequence-

matching techniques is useful for the purposes of time-series trend matching.

4.2 The Scoring Matrix

A scoring matrix operates by assigning a pre-determined value to each character aligned to another character in a string, then calculating a score for the total match based on the sum of each character score in the alignment. Table 4.1 shows the BLOSUM62 scoring matrix for protein substitution, commonly used in bioinformatics. In the example, the string -C-G-A- would match to the string -P-G-C- with a score of ((-3) + 6 + 0) = 3. This score can be used to describe quantitatively the closeness of a match between two strings.

The BLOSUM62 scoring matrix is based on the statistical probabilities for amino-acid substitutions in protein sequences. It is however possible to calculate a similar scoring matrix for quantised time-series data directly during the Symbolic Aggregate Approximation (SAX) procedure, based on the assumption that all characters should be equiprobable (Chiu et al., 2003). The generation of scoring matrices with SAX procedure is examined in detail in Section 5.9

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1

Table 4.1: The BLOSUM62 matrix for protein sequence matching (Setubal & Braeuning, 2006)

4.3 The Smith-Waterman Algorithm (Smith & Waterman, 1981)

The Smith-Waterman matching technique is based on calculating the minimum number of insertions or deletions to transform one string into another. This number of mutational events is a representation of the distance between two strings, similar to the Levenshtein distance (Levenshtein, 1966) (discussed further in Section 6.1). The basic form of the

algorithm and the purpose for which it is used is a simple and biologically relevant example of dynamic programming in action (Eddy, 2004).

For two sequences $A = a_1a_2...a_m$ and $B = b_1b_2...b_m$, a similarity score $U(a, b)$ is given between the sequence elements a and b. A similarity score is given by the appropriate value in a scoring matrix, such as those in Section 4.2. Deletions of length k are given weight W_k. To find pairs of segments with high similarity, we set up a matrix H. The initial values are given by Equation 4.1.

$$H_{k,0} = H_{l,0} = 0 \text{ for } 0 \leq k \leq n \text{ and } 0 \leq l \leq m \tag{4.1}$$

Preliminary values of H have the interpretation that $H_{i,j}$ is the maximum similarity of two segments *ending* in a_i and b_i respectively. These values are obtained from the relationship given in Equation 4.2

$$H_{i,j} = \max(H_{i-1,j-1} + U(a_i, b_i), \max_{k\geq 1}[H_{i-k,j} - W_k], \max_{l\geq 1}[H_{i,j-1} - W_l]) \\ 1 \leq i \leq n \text{ and } 1 \leq l \leq m \tag{4.2}$$

The formula for $H_{i,j}$ follows by considering the possibilities for ending the segments at any a_i and b_j. If a_i and b_j are associated, the similarity is $H_{i-1,j-1} + U(a_i, b_j)$. If a_i is at the end of a deletion of length k, the similarity is $H_{i-k,j} - W_k$. If b_i is at the end of a deletion of length l, the similarity is $H_{i-k,j} - W_i$. Finally, a zero is included to prevent calculated negative similarity, indicating no similarity up to a_i and b_j.

The pair of segments with maximum similarity is found by first locating the maximum element of H. The other matrix elements leading to this maximum value are then sequentially determined with a traceback procedure ending with an element of H equal to zero. This procedure identifies the segments as well as produces the corresponding alignment. The pair of segments with the next best similarity is found by applying the traceback procedure to the second largest element of H not associated with the first traceback. Table 4.2 shows an example of the procedure.

4.4 FASTA and FASTP

The initial FASTP (Fast-Protein) algorithm was developed by Lipman and Pearson in 1985 (Lipman & Pearson, 1985). When it was created, it improved substring searching in databases to the extent of reducing an 8-hour query to 5 minutes of computation on an identical computer. The major improvement was due to the algorithm screening sequences for similarity by looking for aligned similar amino acids before searching the entire database for poor matches (Lipman & Pearson, 1985). The FASTP algorithm

	Δ	C	A	G	C	C	U	C	G	C	U	U	A	G
Δ	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
A	0.0	0.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.7
U	0.0	0.0	0.0	0.7	0.3	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.7
G	0.0	0.0	0.0	**1.0**	0.3	0.0	0.0	0.7	1.0	0.0	0.0	0.7	0.7	1.0
C	0.0	1.0	0.0	0.0	**2.0**	1.3	0.3	1.0	0.3	2.0	0.7	0.3	0.3	0.3
C	0.0	1.0	0.7	0.0	1.0	**3.0**	1.7	1.3	1.0	1.3	1.7	0.3	0.0	0.0
A	0.0	0.0	2.0	0.7	0.3	**1.7**	2.7	1.3	1.0	0.7	1.0	1.3	1.3	0.0
U	0.0	0.0	0.7	1.7	0.3	1.3	**2.7**	2.3	1.0	0.7	1.7	2.0	1.0	1.0
U	0.0	0.0	0.3	0.3	1.3	1.0	2.3	**2.3**	2.0	0.7	1.7	2.7	1.7	1.0
G	0.0	0.0	0.0	1.3	0.0	1.0	1.0	2.0	**3.3**	2.0	1.7	1.3	2.3	2.7
A	0.0	0.0	1.0	0.0	1.0	0.3	0.7	0.7	2.0	3.0	1.7	1.3	2.3	2.0
C	0.0	1.0	0.0	0.7	1.0	2.0	0.7	1.7	1.7	3.0	2.7	1.3	1.0	2.0
G	0.0	0.0	0.7	1.0	0.3	0.7	1.7	0.3	2.7	1.7	2.7	2.3	1.0	2.0
G	0.0	0.0	0.0	1.7	0.7	0.3	0.3	1.3	1.3	2.3	1.3	2.3	2.0	2.0

Table 4.2: The H matrix generated from the application of Equation 4.2 to the sequences A-A-U-G-C-C-A-U-U-G-A-C-G-G and C-A-G-C-C-U-C-G-C-U-U-A-G. The bold elements indicate the trackback path from the maximal element 3.30 (Smith & Waterman, 1981)

was improved; its successor, FASTA (Fast-All), is more sensitive to multiple regions in a particular sequence. This means that the sequence-screening process operates more efficiently and accurately (Pearson & Lipman, 1988). FASTA had another notable improvement, in that it was able effectively to handle local string alignments. This made FASTA a more flexible and therefore powerful searching tool. The method by which FASTA operates is shown in Figure 4.1 (sourced from Bordoli, 2003 (Bordoli, 2003)).

The steps taken are as follows to match string A to string B:

1. Localise a given number of the best regions of similarity between the two sequences. Each identity between two words (k-tuple matches) is represented by a single point. Adjacent word matches will appear as lines as in Figure 4.1

2. Use a scoring matrix to compute a score for the best localised regions. The smaller the word length (k) the faster the search will be, but the less sensitive it will be.

3. Find the best combination of matched localised regions, compute a score for each combination.

4. For all combinations that score above a given threshold, apply dynamic programming to evaluate an optimal score for the overall combination, and list the best combined matches to a sequence. Although the FASTA algorithm is fast, and may be applicable to time-series data matching, it has been replaced by the BLAST algorithm for bioinformatics purposes.

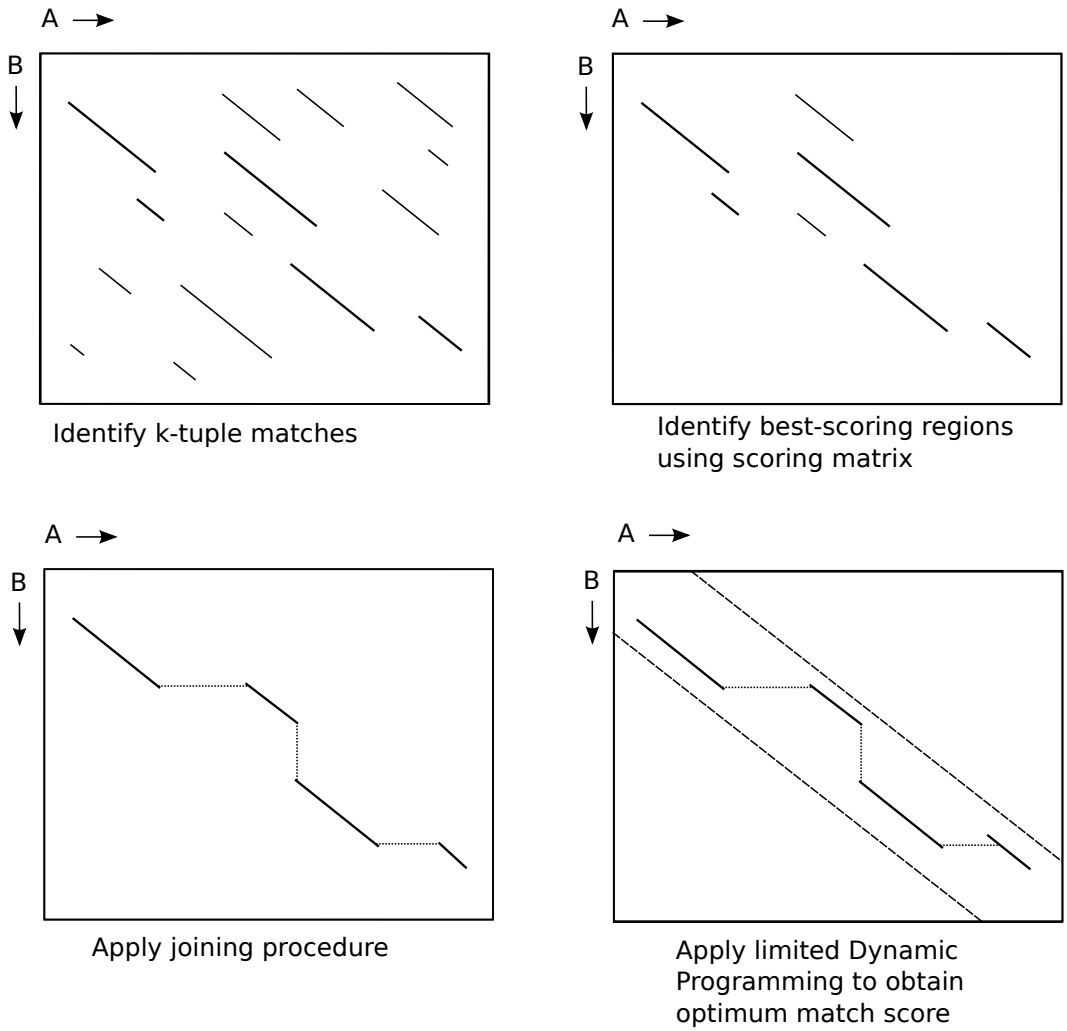

Figure 4.1: The FASTA algorithm for similarity searches (Bordoli, 2003)

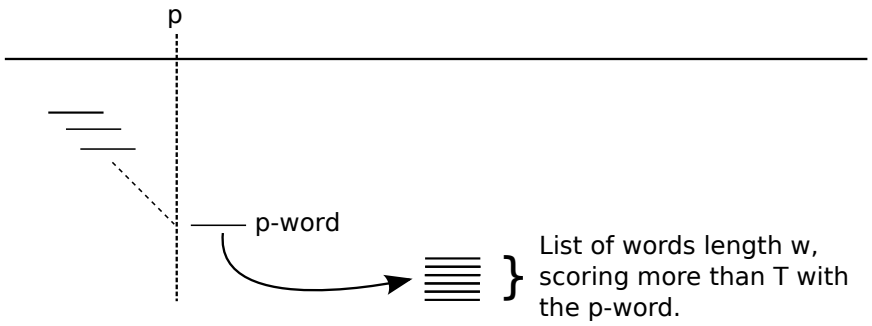

Figure 4.2: BLAST procedure, step one: for each position p of the query, find the list or words of length w scoring more than T when paired with the word starting with p (Bordoli, 2003)

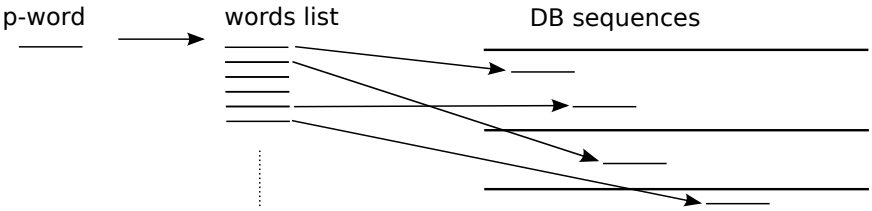

Figure 4.3: BLAST procedure, step two: for each words list, identify all exact matches with database sequences (Bordoli, 2003)

4.5 The Basic Local Alignment Search Tool and PSI-BLAST

The BLAST tool was an order of magnitude faster than those of a similar sensitivity at the time it was developed (Altschul et al., 1990). It operates as shown in Figures 4.2, 4.3 and 4.4 (Bordoli, 2003).

In the diagrams, T refers to a threshold value for matching obtained from a scoring matrix. S refers to the cumulative score obtained from another scoring matrix as the high-scoring element from step 2 is extended. X refers to the maximum allowable drop from maximum S that is permitted before the extension is stopped.

Figure 4.5 demonstrates what is meant by the variables S and X.

BLAST can report e-values, which are an indication of the probability that a sequence

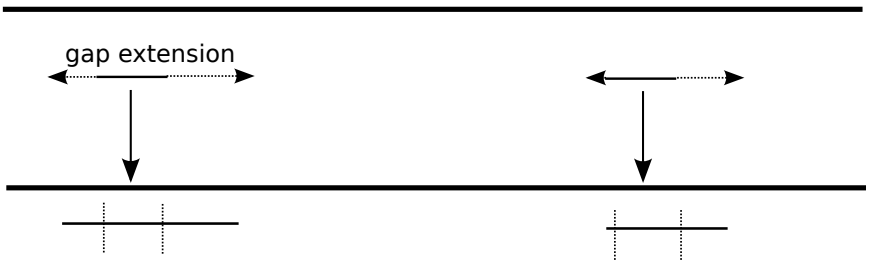

Figure 4.4: Final step of the BLAST algorithm: for each word match (hit), extend the ungapped alignment in both directions. Stop when S decreases by more than X from the highest value reached by S (Bordoli, 2003).

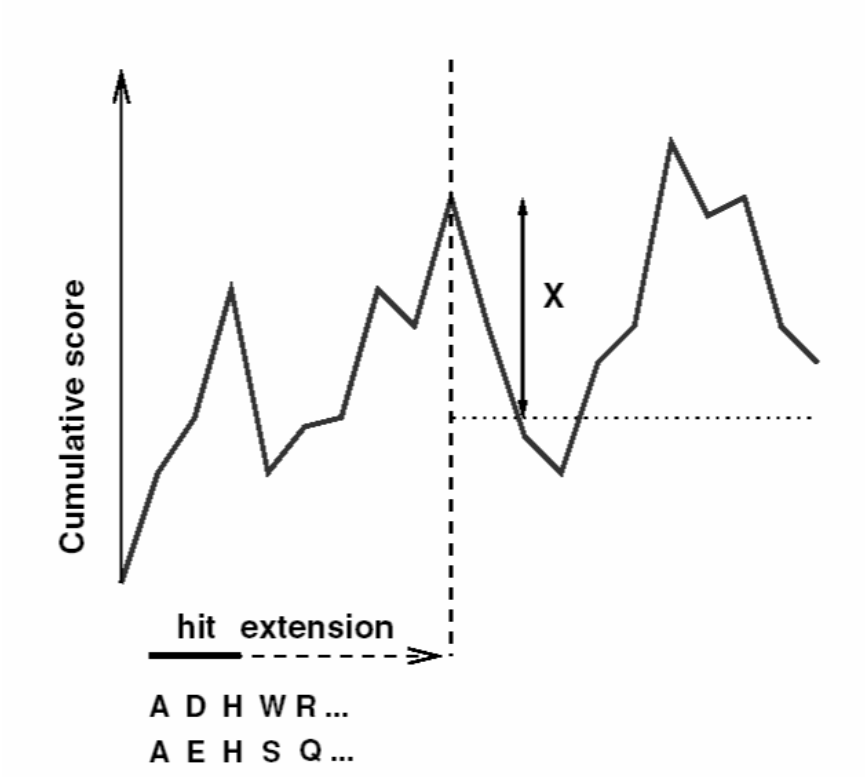

Figure 4.5: Illustration of the final step of the BLAST algorithm, showing cumulative score (S) as a sequence is extended until S drops by the value X (Bordoli, 2003).

match with a given S could have occured by chance. This is accurate only if the scoring matrix used is correct in terms of substitution probabilities (Korf et al., 2003).

A great benefit of the BLAST algorithm is its ability to return multiple matches. Any local match which cannot be extended or shortened without reducing S is considered a locally maximal segment pair, BLAST is able to return all of these pairs above a certain bitscore in one run (Altschul et al., 1990).

The BLAST algorithm has been refined a number of times since it was first implemented. The latest version in use for protein sequences as of 2003 is the PSI-BLAST algorithm (Bordoli, 2003). The improvements made to BLAST in PSI-BLAST include benefits to computing time and sensitivity, as well as the added feature of gap-tolerance (Altschul et al., 1997). The gap-tolerance of PSI-BLAST allows for matching of less precise sequences, and is therefore considered a better algorithm for a more robust trend-matching procedure.

4.6 HMMER

Profile hidden Markov models are also used for gene-finding (Fonzo et al., 2007). The most recent implementation of Hidden Markov Model-based sequence matching is HMMER3 (Eddy, 2011). It promises to be of comparable speed to the BLAST algorithm,

while more sensitive. The major advantage of HMMER over BLAST is that HMMER allows for a more powerful and formal score statistic, with an additional confidence interval which can be calculated for each alignment (Eddy, 2011).

Although HMMER2 was approximately 100-1000 times slower than BLAST-based methods (Eddy, 2011), HMMER3 is only slightly slower than BLAST for many cases. However, it requires training sets to train the profile HMM model for non-biological systems. Due to the release of HMMER3 only after the scope of this study was defined, and the requirement of training sets, it was not possible to compare HMMER-based methods to BLAST based methods in this dissertation. The advantages to using HMMER over BLAST for our purposes are therefore not clear at this point, but it is suggested that HMMER be investigated further in future work, since it may lead to more accurate matches.

The methods employed in bioinformatics operate on string sequences, rather than continuous data. For this reason, filtering and quantisation are requred to convert time-series data into strings for matching.

CHAPTER 5

DISCRETISATION AND DIMENSIONALITY REDUCTION

5.1 Definitions

It is necessary to convert continuous data from an analog chemical process into discrete data which can be handled by digital machines before any of the methods discussed above can be used. This is because the memory of any physical computer is finite, but the range of values possible in continuous data is *infinite*, comprising the entire subset of real numbers.

Discretisation refers to the procedure by which analog data is converted to digital data which can be stored in memory. In this dissertation, time-series data is quantised in the time axis by fixed-interval sampling, and process values are stored to *floating-point* or 32-bit precision.

Quantisation refers in general to converting continuous variables or features into nominal, or *quantized* features. This can also be used to describe certain ranges of continuous process values as a single character, such as in the SAX procedure. This is discussed further in Section 5.9.

Dimensionality reduction refers to techniques which decrease the amount of memory required to represent a given dataset. Note that if a dataset is either quantised or dimensionally reduced, some information will be lost, and the original dataset cannot be reconstructed from a dimensionally reduced one. This property distinguishes dimensionality reduction from *data compression*, since information is irretrievably lost.

5.2 Shapelets

Time series shapelets (Ye & Keogh, 2009) are used in image recognition, and rely on libraries of stored prominent motifs found within time-series data. Matches are classified using decision trees which compare subsequences that are maximally representative of a class.

Shapelets are selected based either on learning sets, or by using knowledge of the process in question to manually build decision trees. Process data may not necessarily be classified by this method appropriately; online chemical processes do not always have large enough learning sets to ensure accurate shapelet identification. Due to the unpredictable nature of process disturbances, operator knowledge of which motif to select as a relevant shapelet may be incomplete.

5.3 Wavelets

It has been shown that Haar wavelet transforms (HWT) can outperform discrete Fourier transforms (DFTs) when reducing dimensionality in time series (Chan & Fu, 1999). Wavelets have the helpful multiresolution property, but they are only defined for time series which are an integer power of two in length (Lin et al., 2003). However, it is possible in practice to pad a time-series in order to make it the next largest power of two in length.

One important feature of wavelets and DFTs is that they are real valued. This limits the algorithms, data structures and definitions available for them. In anomaly detection, the probability of observing any particular set of wavelet coefficients cannot be meaningfully defined, since the probability of observing any real number is zero (Lin et al., 2003). For this reason, HWTs and DFTs are not considered for dimensionality reduction in this work.

5.4 Triangular Episodic Representation

Complete, correct, robust and compact models can be constructed using a small base of primitive representations for process behaviours (Cheung & Stephanopoulos, 1990). The qualitative representations can be defined by any time over which the qualitative state of the process variable x is constant. This qualitative state is defined as in Equations 5.1 to 5.4.

$$\mathrm{QS}(x,t) = \begin{cases} \text{undefined if } x \text{ is discontinuous at } t \\ < [x(t)], [\partial x], [\partial^2 x] > \text{elsewhere} \end{cases} \tag{5.1}$$

Where:

$$[x(t)] = \begin{cases} + \text{ if } x > 0 \\ - \text{ if } x < 0 \\ 0 \text{ if } x = 0 \end{cases} \tag{5.2}$$

$$[\partial x(t)] = \begin{cases} + \text{ if } \partial x > 0 \\ - \text{ if } \partial x < 0 \\ 0 \text{ if } \partial x = 0 \end{cases} \tag{5.3}$$

$$[\partial^2 x(t)] = \begin{cases} + \text{ if } \partial^2 x > 0 \\ - \text{ if } \partial^2 x < 0 \\ 0 \text{ if } \partial^2 x = 0 \end{cases} \tag{5.4}$$

This triangular episodic representation (TER) gives seven basic types of episodes that can describe an interval of process behaviour (Kivikunnas, 1998). Figure 5.1 demonstrates the seven types (sourced from Kivikunnas (1998), adapted from Cheung & Stephanopoulos (1990) and Cheung (1992)). Converting time-series intervals into character strings as defined in Figure 5.1 would quantise *and* dimensionally reduce representation of the time-series. These character strings could then be used by string-matching algorithms.

However, noisy signals would have to be filtered and segmentation is suggested before this algorithm could be applied to time series in the original form. These requirements have a detrimental effect on the computational efficiency of this method of dimensionality reduction, but may not preclude the use of TER in online motif-matching.

5.5 The Bottom-Up Segmentation Algorithm

Bottom-Up segmentation begins by creating the finest possible approximation of the time series, so that $n/2$ segments are used to approximate the n- length time series. The cost of merging each pair of adjacent segments is calculated, and the algorithm begins to iteratively merge the lowest cost pair until a stopping criteria is met (Keogh et al., 2001b).

This technique cannot handle online or streaming data, so for online use a modified version was developed, the Sliding Window And Bottom-Up algorithm (SWAB).

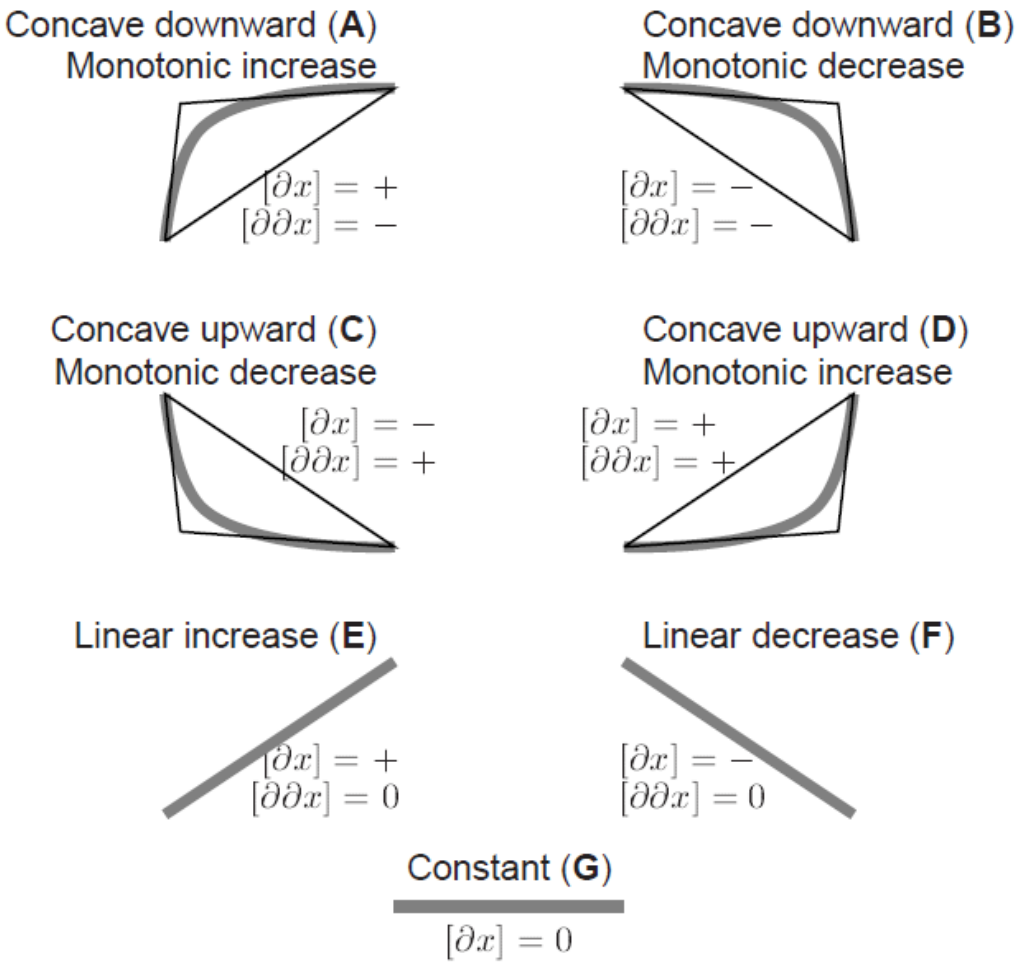

Figure 5.1: Triangular episodic representation. Seven basic types of episodes used for interval description. Each episode type is denoted by a letter from the set {A, B, C, D, E, F, G} (Kivikunnas, 1998)

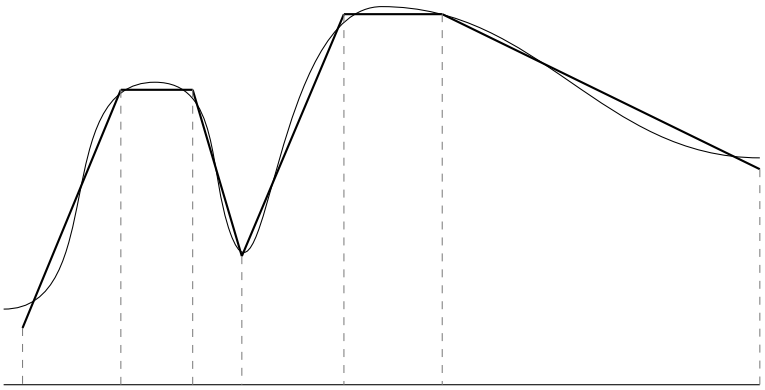

Figure 5.2: An illustration of a Piecewise Linear Approximation to a time series dataset. Note that straight lines are used to approximate continuous curves in this diagram, but may also be used to approximate discrete data

5.6 The Sliding Window and Bottom-Up Algorithm (Keogh et al., 2001b)

The SWAB algorithm keeps a small buffer that is initially chosen so that there is enough data to create about 5 or 6 segments. Bottom- Up segmentation is applied to the data in the buffer and the leftmost segment is reported. The data corresponding to this segment are then removed and more datapoints are read into the buffer. For small buffer sizes, this method converges to the Sliding Window segmentation approach, for large buffers, it approximates the bottom-up segmentation (Keogh et al., 2001b).

The SWAB method combines the online nature of Sliding Window segmentation with the improved accuracy and speed of the Bottom-Up algorithm (Keogh et al., 2001b).

5.7 Piecewise Linear Approximation (Segmentation)

Segmentation refers to the representation of time series in a dimensionally reduced sequence of approximations. These segments can be curves (Arora & Khot, 2003), but more commonly they are represented by straight lines. This case is also known as a piecewise-linear approximation (PLA) (Keogh et al., 2001b; Pavlidis & Horowitz, 1974). Figure 5.2 demonstrates what is meant by a piecewise-linear approximation.

PLA can also be used as a post-processing step to filter spurious matches found using Symbolic Aggregate Approximation (SAX). This technique is explored in Hung & Anh (2007), but due to time constraints is not investigated in this dissertation.

The SAX procedure makes use of another technique of dimensionality reduction, namely Piecewise Aggregate Approximation (PAA).

5.8 Piecewise Aggregate Approximation

In order to reduce the search space, it is often required to resample an original time series X of length n to a reduced length m. A simple approach is to divide the entire time range into blocks and average over the blocks, as in Equation 5.5 (Keogh et al., 2000; Chiu et al., 2003). This approach is at face value similar to the boxcar filter, however it reduces dimensionality while the boxcar filter does not.

$$\bar{x}_i = \frac{m}{n} \sum_{j=\frac{n}{m}(i-1)+1}^{\frac{n}{m}i} x_j \tag{5.5}$$

PAA has fixed time windows in which it operates. A variant of PAA known as APCA (Adaptive Piecewise Constant Approximation) has also been developed (Keogh et al., 2001a). While this method provides greater accuracy when reconstructing time-series datasets from the dimensionally-reduced version, the additional complexity introduced by indexing the length of each segment will likely make it less robust when combined with the BLAST algorithm.

5.9 Symbolic Aggregate Approximation (SAX)

These continuous values need be quantized into character strings. The Symbolic Aggregate approXimation (SAX) method (Lin et al., 2003) uses breakpoints which will result in an equal probability of letters if the data were normally distributed. This is a highly desirable characteristic for sequence-matching algorithms as it makes the scoring matrices easy to calculate. Although the full procedure is outlined in (Chiu et al., 2003) and (Lin et al., 2003), Figure 5.3 illustrates the basic concept of SAX: using the average value for a window of time series data to allocate a specific character.

PAA may achieve similar results to the more complex and computationally demanding methods such as fourier transforms and wavelets (Chiu et al., 2003), so it and SAX are used as the baseline technique for converting time-series data into character strings.

Breakpoints are a sorted list of numbers $\beta_0 \ldots \beta_n$ such that the area under the Gaussian curve for the data set is equal for every β_i to β_{i+1}. The reason for the varying size of the distances used to quantise approximations of the data is that process data tends to be normally distributed. This implies that in order to make each character equally likely, the breakpoints must be calculated using an assumption of normally-distributed process data (Chiu et al., 2003). In practice, breakpoint values can be obtained by sorting an adequately large sample of the dataset and using interpolation to determine values for which each region will be approximately equiprobable, as in Figure 5.4.

Making each character equally likely maximises the amount of *Shannon entropy* (the expected value of information) of any sequence generated by SAX.

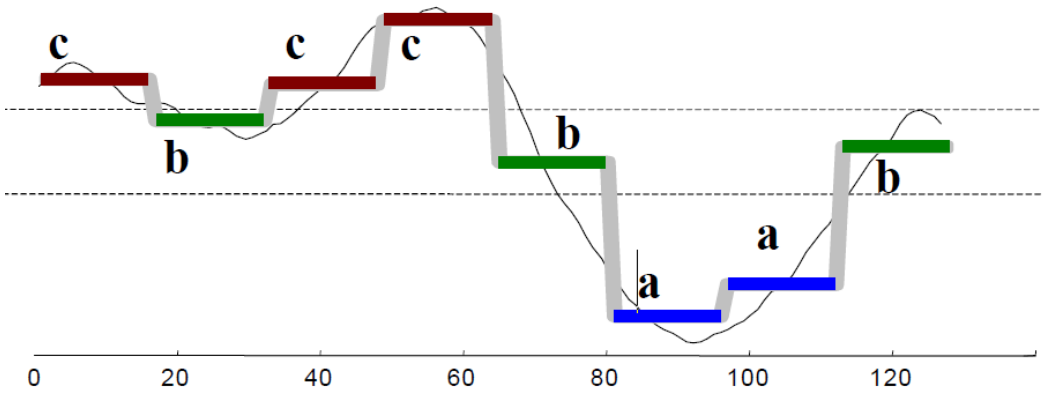

Figure 5.3: Illustration of the PAA and SAX technique, note that breakpoints are normally distributed (Chiu et al., 2003)

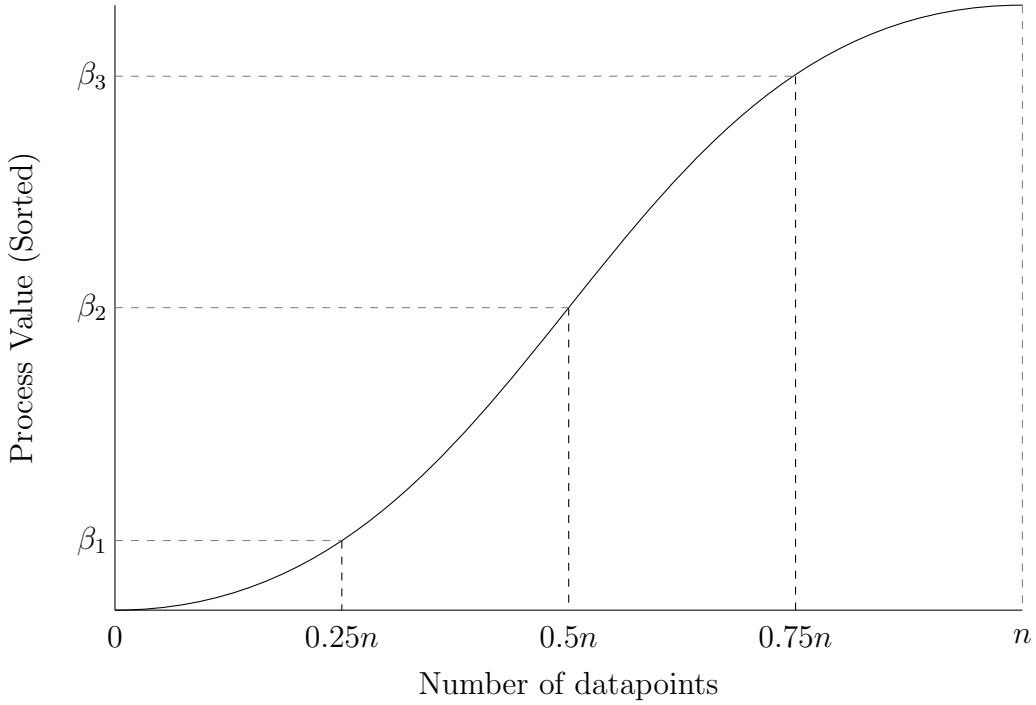

Figure 5.4: m Breakpoints are calculated by sorting the database time-series vector and interpolating m linearly spaced values in the vector, excluding the endpoints.

The SAX procedure also outlines a method to calculate similarity scores for each assigned character. If β_i represents a breakpoint for a given character, Equation 5.6 can be used to generate a similarity scoring matrix (Lin et al., 2003).

$$\text{score}_{i,j} = \begin{cases} 0 & \text{if } |i-j| \leq 1 \\ \beta_{\max i,j-1} - \beta_{\min i,j} & \text{otherwise} \end{cases} \tag{5.6}$$

The scoring matrix generated during the SAX procedure could be used in the BLAST algorithm, as a replacement for matrices such as the BLOSUM62 scoring matrix.

Both SAX and PSI-BLAST have statistical measures for quantifying the predicted accuracy of a match. However, these measures are directly optimised by their respective algorithms, which could potentially lead to unfair comparisons. It is necessary to investigate other methods for quantifying accuracy.

CHAPTER 6

MEASURES FOR MATCHING ACCURACY

When discussing the similarity of two time-series, it is necessary to determine a method of *quantifying* this similarity. Once quantified, similarity scores can be used to compare the accuracy of matches found by different motif-matching techniques.

6.1 Levenshtein Distance Measures

The Levenshtein distance (Levenshtein, 1966) between two strings can be described as the minimum number of string edits required to transform one string into the other, based on the assumption that the possible operations are:

- Insertion of a single character
- Deletion of a single character
- Substitution of a single character for another

This method can be used as a metric for describing the similarity of two strings, by using a recursive procedure as outlined in Equation 6.1.

$$\mathrm{Lev}_{a,b}(i,j) = \begin{cases} 0 & \text{if } i = j = 0 \\ i & \text{if } j = 0 \text{ and } i > 1 \\ j & \text{if } i = 0 \text{ and } j > 1 \\ \min \begin{cases} \mathrm{Lev}_{a,b}(i-1,j) + 1 \\ \mathrm{Lev}_{a,b}(j,i-1) + 1 \\ \mathrm{Lev}_{a,b}(i-1,j-1) + [a_i \neq b_j] \end{cases} & \text{otherwise} \end{cases} \tag{6.1}$$

The Levenshtein distance measure has the limitation that it is only defined for character strings, and therefore cannot be used in this form to compare methods which do not employ quantisation. For this reason, Levenshtein distance is not an appropriate measure for use in this dissertation.

6.2 Euclidean Distance Techniques

The Euclidean distance between any two points in a 2-dimensional plane (also known as the Pythagorean distance) is defined by Equation 6.2 (Deza & Deza, 2009).

$$\|x - y\|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{6.2}$$

One metric that can be used to quantify the difference between two time-series (X and Y) is the Euclidean distance between them, which is defined for time series as in Equation 6.3 (Keogh et al., 2000).

$$D(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{6.3}$$

This measure may only be directly applied to time-series of the same length.

In specific cases, it may be beneficial to use weighting to increase the relative importance of certain sections of a time-series motif (Keogh et al., 2000). A weighted Euclidean distance between two time-series can be expressed as in Equation 6.4 (Keogh et al., 2000).

$$DW([X, W], Y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \tag{6.4}$$

Euclidean distance does not allow for two time-series of differing lengths to be compared. Since process motifs can often be distored in the time axis and represent the same behaviour, another metric for accuracy should be used.

6.3 Dynamic Time Warping

Dynamic Time Warping (Sakoe & Chiba, 1978) is often used as a faster and more robust method than Euclidian Distance to quantify the similarity of two time series (J & Pazzani, 2000). It (DTW) uses a dynamic programming approach to align two time series and a specific word template so that a distance measure is minimised (Berndt & Clifford, 1994).

If there are two time series, x and y of length m and n respectively, DTW can align the two sequences by constructing an m-by-n matrix Z where each element $Z_{i,j}$ contains

the squared Euclidean distance between the points x_i and y_j. A warping path Q is the contiguous set of C matrix elements that defines a mapping between x and y, such that:

$$\begin{aligned} & Q = q_1, q_2 \dots q_c \dots q_C \\ \text{where} \quad & q_c = (i,j)_c v \\ \text{and} \quad & \max(m,n) \leq C < m+n-1 \end{aligned} \tag{6.5}$$

The warping path is subject to some constraints (Ratanamahatana & Keogh, 2004):

- It must start and finish in diagonally opposite corner cells of the matrix Z. Therefore, $q_1 = (1,1)$ and $q_C = (m,n)$.
- It must be continuous and monotonic, including every row and column of Z. Given $q_c = (i,j)$ and $q_{c-1} = (i',j')$, it follows that $0 \leq i - i' \leq 1$ and $0 \leq j - j' \leq 1$

There are exponentially many warping paths that satisfy these constraints, so the DTW distance between two time-series is defined as:

$$\text{DTW}(x,y) = \min \sqrt{\sum_{c=1}^{C} Z(q_c)} \tag{6.6}$$

Dynamic programming is used to calculate this optimal warping path. To improve computational efficiency, modern implementations of DTW employ a lower bounding technique based on a warping window (envelope). The most commonly used warping constraints are the Sakoe-Chiba band (Sakoe & Chiba, 1978) and the Itakura parallelogram (Itakura, 1975; Ratanamahatana & Keogh, 2004). The DTW algorithm implemented in this way scales linearly with the size of both time-series, and so has an order $O(mn)$ (Ratanamahatana & Keogh, 2004). This order notation is explained in Section 3.1

DTW can also be used as a method to match subsequences (Berndt & Clifford, 1994), as implemented in the Machine Learning Python Library (`mlpy`) (Albanese et al., 2012). The `mlpy` implementation of DTW was selected as the baseline technique for time-series motif-matching, since there is a large amount of literature which employs DTW and it is popular in several applications, including speech and image recognition (Sakoe & Chiba, 1978).

Since DTW will be used as a motif-matching technique in this dissertation, additional metrics for match accuracy are required.

6.4 Edit Distance On Real Sequence (EDR)

The EDR algorithm operates on discrete data, and was developed for trajectory tracking (Chen et al., 2005). It is noise-tolerant, gap-tolerant, and provides a similar matching performance to DTW while being less computationally intensive (Chen et al., 2005). If $Rest(X)$ is defined as the time-series X excluding the first entry x_1, then the EDR algorithm operates between two time-series of length n and m as in Equations 6.7 and 6.8 (Chen et al., 2005). These equations have been modified for the 1-D case; EDR can be applied to multidimensional datasets just as easily.

$$match = \begin{cases} 1 & \text{if } |x_1 - y_1| > \epsilon \\ 0 & \text{if } |x_1 - y_1| \leq \epsilon \end{cases} \tag{6.7}$$

$$\text{EDR}(X, Y) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ \min \begin{cases} \text{EDR}(Rest(X), Rest(Y)) + match \\ \text{EDR}(Rest(X), Y) + 1 \\ \text{EDR}(X, Rest(Y)) + 1 \end{cases} & \text{otherwise} \end{cases} \tag{6.8}$$

EDR can be used to determine the similarity of two time-series. Due to the binary nature of *match* in Equation 6.7, EDR does not impose harsh penalties for outlier data, which makes it more robust when data quality is questionable.

6.5 Sequence Weighted Alignment (SWALE)

The EDR technique penalises gaps and mismatches, but does not reward matches. The Sequence Weighted Alignment technique (SWALE) addresses this problem by modifying the EDR approach to allow for weighting of gap penalties against a reward for a minimum ϵ match (Morse & Patel, 2007). Equation 6.9 shows how the SWALE algorithm improves on EDR, given that gap_c represents the gap penalty and $reward_m$ represents the reward for matching sequences (Morse & Patel, 2007). The relationship between the two weights can be obtained using training datasets, or tuned by hand.

$$\text{SWALE}(X, Y) = \begin{cases} n \times gap_c & \text{if } m = 0 \\ m \times gap_c & \text{if } n = 0 \\ reward_m + \text{SWALE}(Rest(X), Rest(Y)) & \text{if } match = 0 \\ \min \begin{cases} \text{SWALE}(Rest(X), Y) + gap_c \\ \text{SWALE}(X, Rest(Y)) + gap_c \end{cases} & \text{otherwise} \end{cases} \tag{6.9}$$

In both EDR and SWALE, the recommended error threshold (ϵ) is half the standard deviation for the time-series dataset (0.5σ) (Morse & Patel, 2007).

6.6 Fast Time Series Evaluation (FTSE)

The Fast Time Series Evaluation is an algorithmic method which improves on conventional dynamic programming approaches to implement time-series accuracy measures, such as EDR and SWALE. The full algorithm is given in (Morse & Patel, 2007). Due to implementation time constraints, FTSE is not used in this work, but should be considered at a future stage to speed up the SWALE matching used.

Part II

Experiment

CHAPTER 7

EXPERIMENTAL DESIGN

A framework for comparing motif-matching methods in terms of computational load, robustness and selectivity has been developed. Three methods; SAX-PSIBLAST, TER-PSIBLAST and DTW; are tested and used to demonstrate a proof-of-concept for a streaming motif-matching and prediction framework.

7.1 Matching Framework

Figure 7.1 shows how a library of time-series and query data is processed by each algorithm to produce a match with quantifiable accuracy in the ideal case. This method can be used as part of a larger system which predicts future process behaviour based on current process trends. However, due to time constraints, a modified dataflow was used. The modified framework uses an approximate method similar to Triangular Representation, but with fixed time intervals. This modified framework is shown in Figure 7.2.

7.2 Testing

Three matching methods are considered for testing:

1. The SAX procedure, combined with PSI-BLAST string matching.
2. The naive triangular representation of filtered data, combined with PSI-BLAST.
3. Dynamic Time Warping as implemented in the `mlpy` python library.

Three criteria are used to compare each matching method:

1. Computational speed: How quickly can this algorithm return a match? To which order does the computation time scale in terms of Query and Database length?

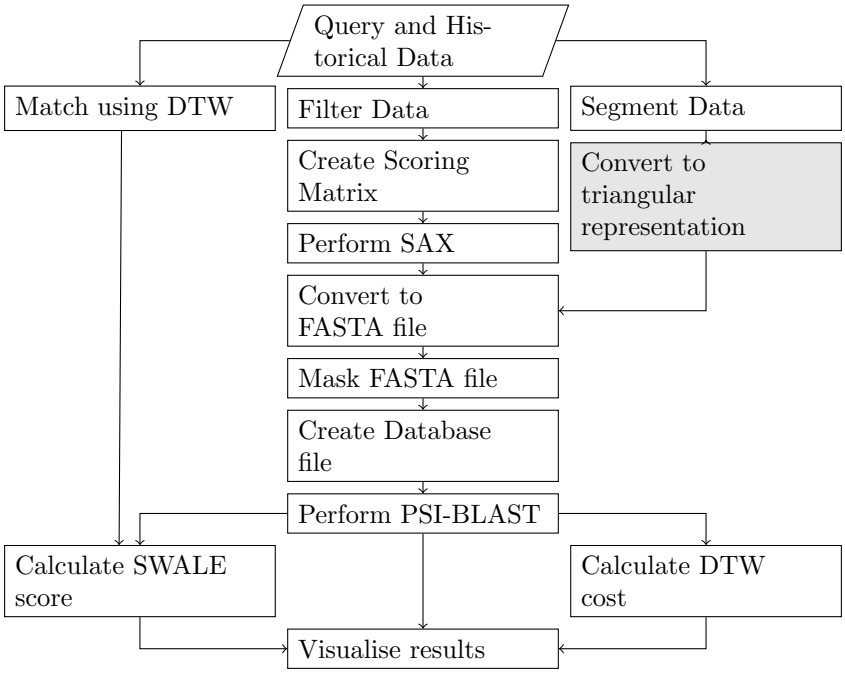

Figure 7.1: A simplified overview of the desired data flow for testing

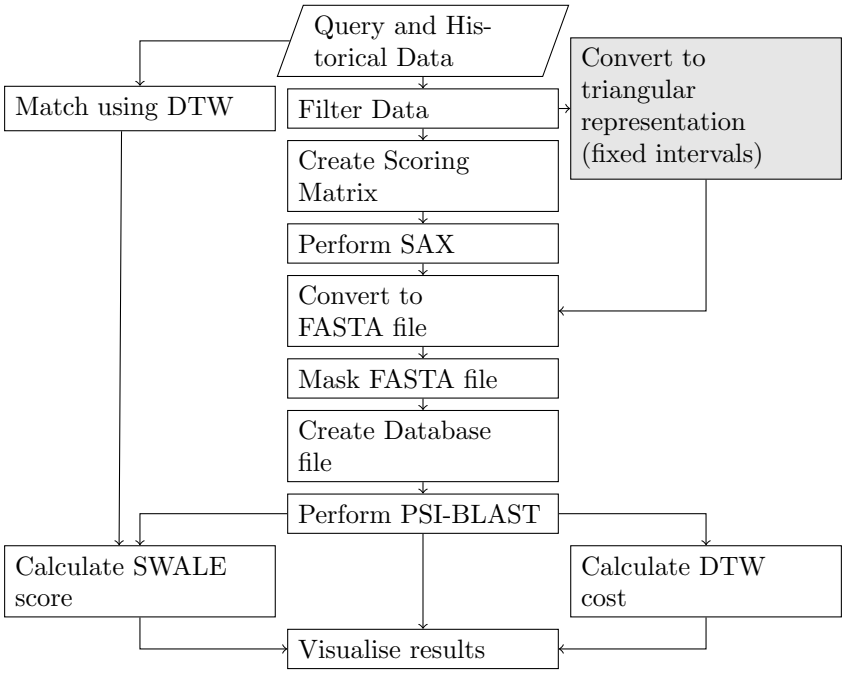

Figure 7.2: An overview of the process coded: Note that segmentation is not included, as during development the implemented segmentation library was deemed too slow

2. Matching accuracy: How often is a match found? How similar is the best match to the original query?

3. Robustness: How much noise can be tolerated? What mutators can be applied to the dataset before matching fails?

7.2.1 Computational Speed

For the SAX and triangular representation methods, there is front-end loading involved in converting time-series data to BLAST databases for matching. The testing procedure measures the wall-time (the time as measured by an external clock, including I/O delays) rather than CPU-time for the creation of these databases because there are hard-disk intensive operations used. In addition to the time used to create the database, the wall-times taken to perform the matches from stored data are measured.

The match time is a combination of the time taken to convert time-series data to strings and the time taken by PSI-BLAST to generate a match. There is expected to be communication overhead between PAA section written in Python and the `psiblast` executable, therefore wall times are used to include this overhead as part of matching time.

Dynamic Time Warping match times include the time taken to reduce dimensionality. This method also utilised wall-times for experimental consistency, but DTW does not require any hard drive operations once the datasets are loaded into memory.

It is expected that both database creation time and matching time will scale with the size of the historical database series, and that match times will scale with query length. However, it is expected that there will be a minimum overhead to the PSI-BLAST based methods, since these methods require communication between the interpreter and `psiblast` executable.

All the time tests are performed on an Intel i5 M430 Dual Core Processor 2.27 Ghz running Linux Ubuntu 12.04 32-bit, using only one core at a time.

7.2.2 Matching Accuracy

In order to quantify the similarity between the queried sequence and its corresponding match, the following metrics are examined where applicable:

- BLAST bitscore
- DTW warping distance
- SWALE score

The DTW score cannot be used as an accuracy metric for all three cases, since it is directly optimised during DTW matching and will skew results in DTW's favour. The BLAST score is representative of the quality of match only after dimensionality has been reduced, meaning it does not directly relate to the original dataset. This leaves SWALE as the only scoring metric examined here that can objectively be used for the test cases.

In terms of accuracy measures, it may be possible to develop a weighted average of these values to create a lumped match score. The creation and testing of this 'lumped score' is outside the scope of this study, but may be useful in future definitions of match accuracy.

Without prior knowledge of the process being examined, there is a continous spectrum of match-quality based on these scores. The thresholds for what constitute a good match must be user-defined in this case, and will be situation-dependent. The testing procedure also allows for a comparison between these methods, if any two methods correlate strongly, they may be considered to be interchangeable. However, this is not expected, as the different scoring metrics are based on different principles.

To get an intuitive feel for how the normalised SWALE score correlates to the dissimilarity of two time series, Figure 7.3 shows SWALE scores for two superimposed datasets at different levels of noise:

Although it is recommended that SWALE score thresholds are 0.5σ for the dataset used (Morse & Patel, 2007), the experiment is performed on several datasets with different standard deviations. If the SWALE scoring method were adjusted for each dataset, it would prevent comparison of SWALE results across different lengths of data. For this reason, a fixed SWALE threshold of 0.05 is used over all datasets, regardless of their individual σ values. It is important to note, however, that the perceived noise tolerance of an algorithm is strongly influenced by the SWALE score threshold in the real-world case, and therefore tuning of the SWALE threshold value may be required for noisy process data which are not pre-filtered.

7.2.3 Robustness

Noise tolerance is considered the primary concern of the matching algorithms tested, as signal noise is a common problem in Control Engineering. Gaussian noise is applied to both the historic time-series database and query sequence, and then a match is performed. The SWALE scores for matches are based on the original non-noisy time-series, since in practice this would represent the actual output values for the process.

In many cases, the potential for 'false positives' during matching would be considered a metric for matching accuracy. This is usually the case when using matches for classification purposes. For the purposes of this dissertation, the criteria for a match to be considered *significant* is chosen such that the match length must be at least 5 datapoints

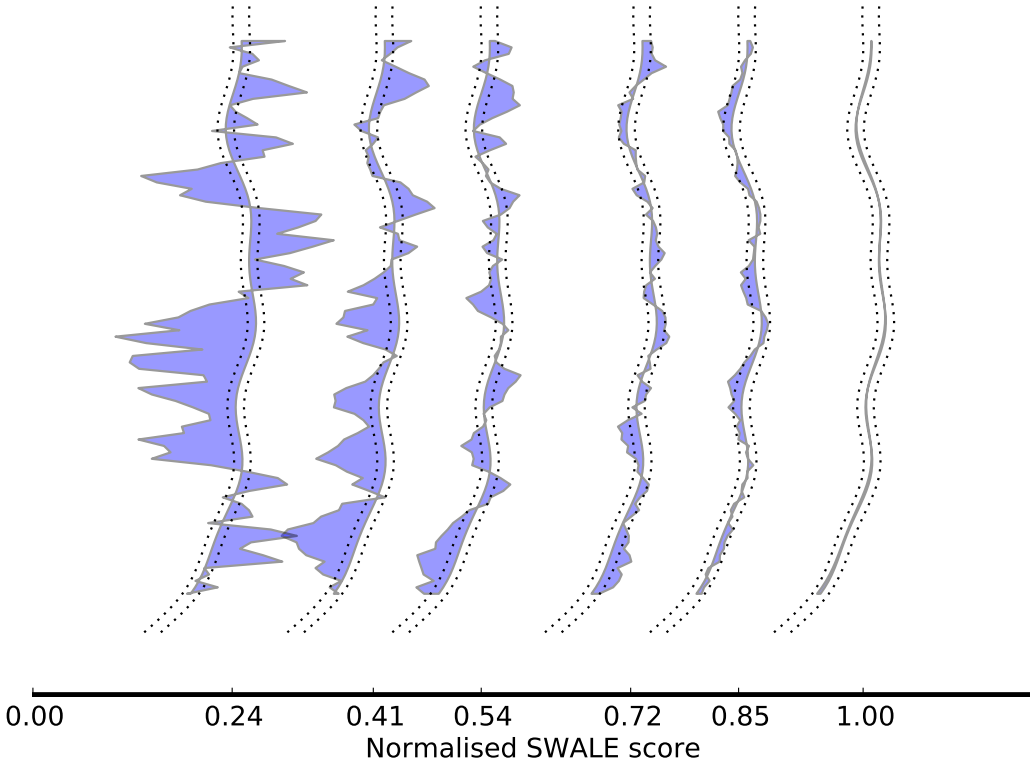

Figure 7.3: As time series are more similar, their normalised SWALE scores range from 0 (completely uncorrelated) to 1 (perfectly correlated). The dotted lines around the original dataset refer to the limits for the SWALE algorithm to consider the series matched.

after dimensionality reduction is applied. Classification and machine learning methods for process prediction are outside the scope of this study, but an *ad hoc* metric for this classification has been developed:

Hit rate The fraction of significant matches which score above a specific SWALE score.

Match rate The number of significant matches found over the total number of queries performed.

Here, the hit rate is used as an indicator of how often a match found is genuine (selectivity), and the match rate is used as an indicator of how good an algorithm is at finding genuine matches (power). The criteria for a significant (non-trivial) match is chosen such that the match length must be at least 5 datapoints after dimensionality reduction is applied, otherwise the match is considered trivial.

It is presumed that in general, match quality (and therefore SWALE scores) will decrease with increasing noise levels, up to a threshold where pure noise would return uncorrelated sequences. It is therefore also assumed that increasing noise will cause a decrease in the match rates and hit rates for a given method.

7.3 Streaming Prediction Framework

A proof of concept is implemented for a streaming prediction framework similar to CQP (Gao & Wang, 2002), but utilising motifs found using DTW, SAX-PSIBLAST, TER-PSIBLAST and a naïve autoregression is developed.

For every method except autoregression, buffered data is used as a query to a database of stored process history and the process behaviour which proceeds from each match in the database is considered a process prediction. PSIBLAST-based methods return multiple matches, and the best matches found according the BLAST scoring are displayed to the operator at levels of transparency based on their rank.

Autoregressive models, on the other hand, are built each time from the buffered data, and then used to predict future behaviour. The reason for the application of this method as opposed to historical data being used to build a process model is because it is intended that the final system be data-independent and prevent implementation overheads when encountering a new process.

As a proof of concept, this program operates offline and is not attatched to any process with streaming data. Time constraints did not allow for an online test of this framework and this test is left for future work.

CHAPTER 8

IMPLEMENTATION

Python 2.7, an interpreted language, has been used. It was selected because it is open-source, has a large number of user-made libraries for scientific purposes, and supports object-oriented programming. Due to the similarity between python code and pseudocode, python is also considered an easy programming language to learn.

The sample data used was the Pseudo Periodic Synthetic Time Series, donated for research by Keogh & Pazzani (1999).

8.1 Synthetic Data Source

The Pseudo Periodic Synthetic Time Series (Keogh & Pazzani, 1999) is generated by Equation 8.1

$$\bar{y} = \sum_{i=3}^{7} 1/2^i \sin 2\pi(2^{2+i} + \text{rand}(2^i))\bar{t} \tag{8.1}$$

Where rand(x) generates a random integer between 0 and x. Although the data appears periodic, it never repeats itself exactly (Keogh & Pazzani, 1999).

8.2 Data Filtering

The `scipy.signal` (Jones et al., 2001–) library is used for filtering when datasets contain significant noise which could affect trend matching. Filtering also forms part of the basic triangular representation string conversion in every case, as noise would be extremely disruptive when determining derivatives over fixed intervals.

8.3 Scoring Matrix Creation

First, the initial historical time-series is sorted to give a monotonically increasing dataset of length n, from which SAX breakpoints are calculated directly as in Figure 5.4. This ensures that for a given historical dataset, each character will be roughly equiprobable. A scoring matrix is generated using these breakpoints as in Equation 5.6 and stored in memory.

This matrix is saved as a BLOSUM62-like scoring matrix which overwrites the BLOSUM62 matrix stored in the PSI-BLAST default path. This was the only recourse, as the NCBI PSI-BLAST executable (Altschul et al., 1997) does not support custom scoring matrices as arguments.

8.4 SAX Implementation

Piecewise Aggregate Approximation (Section 5.8) is used to reduce the dimensionality of data, and the breakpoints calculated from historical data in Section 8.3 are also used for the query.

After this step, both the historical data and query data are represented by character strings.

8.5 Conversion to FASTA file

The python library Biopython (Cock et al., 2009) has built-in functions to build sequences and save them in the FASTA format. This format is used by PSI-BLAST. The nature of the FASTA format, allowing multiple sequences stored in one file with different headers. This allows for the PSI-BLAST process to be extended to performing multiple matches simultaneously, but this functionality is not used in this dissertation.

8.6 Creation of Database

The BLAST+ executable `makeblastdb` was used to convert the historical data sequence into a PSI-BLAST searchable database. This conversion allows for multiple process sequences to be stored in a single database with headers, and could be extended to allow for multiple simultaneous matches. This was not investigated in this dissertation, but care should be taken when storing datasets from multiple sources in one database, as it may cause undesired match results from undesired processes.

8.7 PSI-BLAST matching

The `psiblast` executable was used to perform matches, which were given as outputs written to a csv file. All matches found are listed in order of lowest e-value, with the following information:

- The query database sequence IDs for the match. (In this implementation, there is only one sequence in both, so this information is not used.)
- The length of the match found.
- The number of mismatching characters and gaps.
- The begining and end value of both the query and database sequence.
- The e-value and bitscores for each match, as calculated in the BLAST algorithm. These are as described in Section 4.5

The e-values represented in the matches may not give the correct statistical values, since the scoring matrix developed has not been normalised as it would be for biological matches. However, these values can still act as a match-ranking system which can be supplemented by other matching metrics, such as those described in Section 6.

8.8 Triangular Representation (fixed windows)

Due to the increased complexity of this matching method when applied to variable-length segments, full Triangular Episodic Representation was not implemented.

Instead, a fixed-window approach was used to describe time-series in terms of their first and second derivatives over a fixed interval. These derivatives are expressed only as positive (+1), negative (-1), or approximately zero (0). This led to 9 combinations which represent different characters. The similarity score matrix is calculated as in Equation 8.2.

$$\text{cost}_{i,j} = \frac{\mathrm{d}}{\mathrm{d}t}_i \frac{\mathrm{d}}{\mathrm{d}t}_j Q + \frac{\mathrm{d}^2}{\mathrm{d}t^2}_i \frac{\mathrm{d}^2}{\mathrm{d}t^2}_j \tag{8.2}$$

In this case, Q represents a weighting of first derivative mismatches versus second derivatives, which can be tuned by the user. The *ad hoc* nature of this implementation means that the e-values calculated by PSI-BLAST in this case will not be statistically meaningful. However, matches could still be naively ranked using these values.

8.9 Dynamic Time Warping

The `mlpy` (Albanese et al., 2012) python library contains built-in DTW functions. The `mlpy.dtw_std` and `mlpy.dtw_subsequence` functions are used to score matches and find subsequence matches respectively.

It is not described in the `mlpy` documentation which warping constraints are used. In fact, there may be no warping constraints in this implementation. However, since DTW was used as a baseline technique, and was not the focus of this investigation, it will be assumed that the simplest case (unconstrained warping) is taking place.

8.10 SWALE Scoring

A dynamic-programming approach is used to calculate the SWALE score for the match, as outlined in Section 6.5. FTSE was not implemented due to time constraints. This approach affects only the computation time for SWALE scoring, but does not affect the scores themselves.

8.11 Visualisation

The `matplotlib` library is used to plot matches. A figure consisting of three sub-plots is generated. The first sub-plot is shows the region of historical data which matched the query. The second plot superimposes both this region and the match, and the third plot shows which subsequence of the query gave the match. An example of these visualisations can be found in Appendix A.

The streaming prediction framework also utilises the `matplotlib` library. The visualisation strategy is explained in Section 7.3, and a sample output can be seen in Figure 11.1.

8.12 Object-Orientation

In order to keep the application of these methods modular, an object-oriented approach is used. The objects used by the program are:

Dataset An object which contains time-series data and time-stamps. These can be loaded from or saved to csv files. This class contains mutators to introduce noise, vertical stretching, naive resampling and insertion of new time-series data. The PAA, SAX and Triangular Representation conversions are also contained in this class. This class allows for multiple datasets for a single time-stamp series, but

this functionality is not used in this dissertation. The PAA, SAX, and modified triangular representation conversion methods are contained within this class.

Sequence An object which contains character-string data. There are methods for saving these objects as FASTA files contained in the object. A function converting the sequence object into the original PAA-approximated dataset is also included.

Match An object which contains match data listed in Section 8.7, loaded from a `psiblast` csv output.

TPAdistance An object which calculates and stores scoring matrices for the SAX-PSIBLAST technique.

TriangScore An object which generates and stores the scoring matrix for the modified Triangular Representation technique, as in Section 8.8.

In addition to these objects, there are additional static methods. Methods to calculate the Euclidean Distance, DTW cost, and SWALE similarity score for two datasets are included in the program library. Wrapper methods for `makeblastdb` and `psiblast` are used to pass necessary input arguments to each executable. The `plotmatch` function is used to generate the single-match visualisation, and the `realtimesim` method is used to demonstrate the proof of concept for the streaming prediction framework.

Part III

Results

CHAPTER 9

PROCESSING TIME

9.1 BLAST Database Creation

Figure 9.1 shows the time taken to build databases for time-series of various sizes. Database creation appears to be $O(n)$, but there is significant variation in the time taken. It appears that the use of hard-disk operations for database creation causes this variation, but due to the fast nature of this operation, it should not be a practical problem.

9.2 Matching time

9.2.1 Best-Case Matching

Figures 9.2 and 9.3 show how the time taken to perform matches varies with the sizes of query and database used, with no dimensionality reduction. These cases were performed for a best-case matching situation; the queried time-series was directly embedded in the database. DTW and BLAST-based methods scale similarly with database size, but BLAST-based methods scale better with query size. For this reason, DTW will perform better on large databases with small queries, whereas BLAST will perform better on small databases with longer queries.

It can be seen that DTW matches are returned substantially faster than matches using string conversion and PSIBLAST. This is due to the time taken to convert, rather than the PSI-BLAST algorithm itself, as Figure 9.4 demonstrates. It is evident that this conversion procedure is inefficient. Because Python string objects are not primitive Python types, nor mutable like arrays, their use may be the bottleneck. Improving this bottleneck could possibly make the SAX-PSIBLAST more competitve with DTW in terms of computation time.

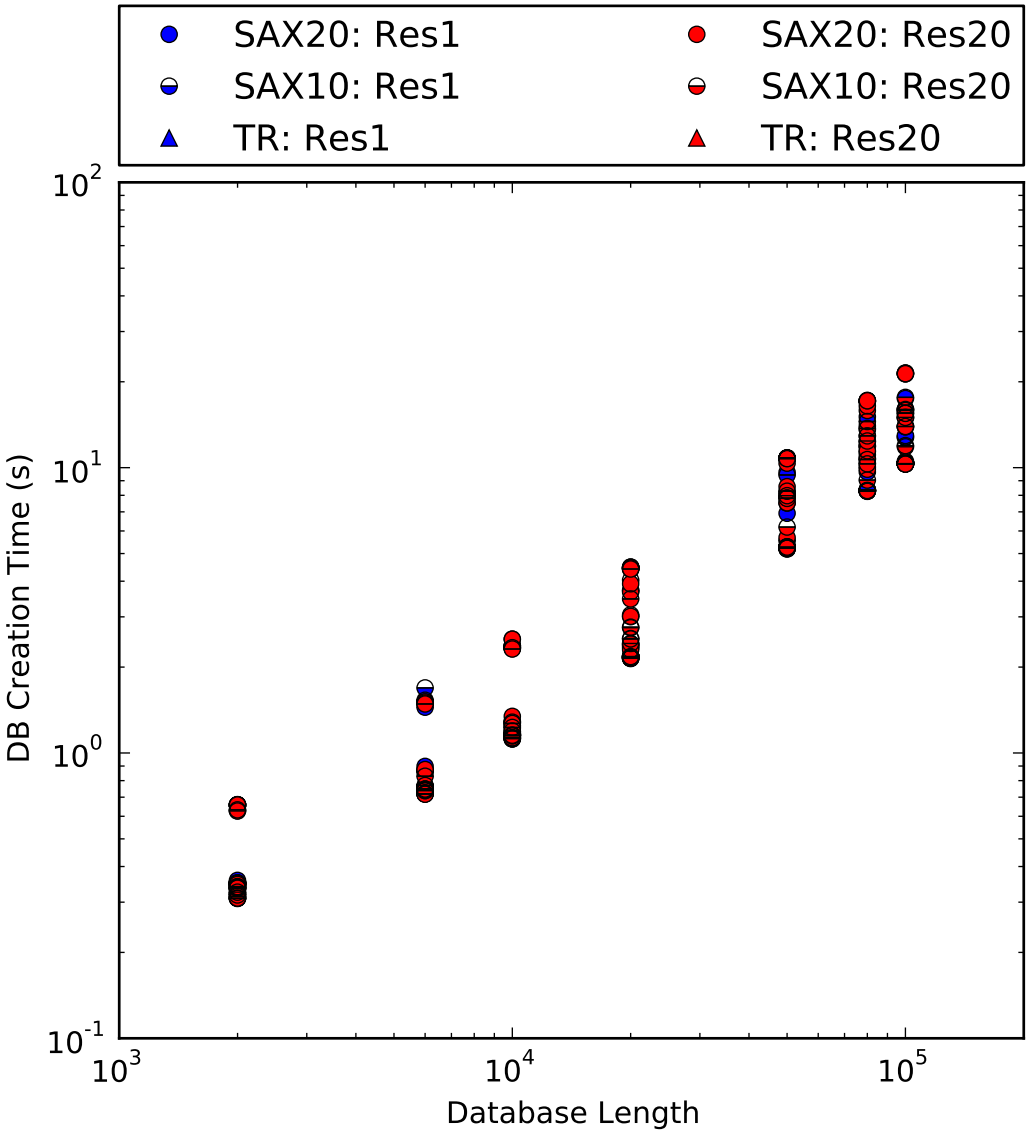

Figure 9.1: Time taken to create BLAST databases from time series, with varying PAA and Triangular Representation time windows. The data is plotted on a log-log axis for clarity, but the trend is linear, not exponential. In this dissertation, Res-i refers to the amount of dimensonality reduction applied; Res1 is without dimensionality reduction, Res20 is a 20-fold reduction. SAX-i refers to the number of breakpoints used during the SAX procedure, eg SAX10 used 10 non-infinite breakpoints.

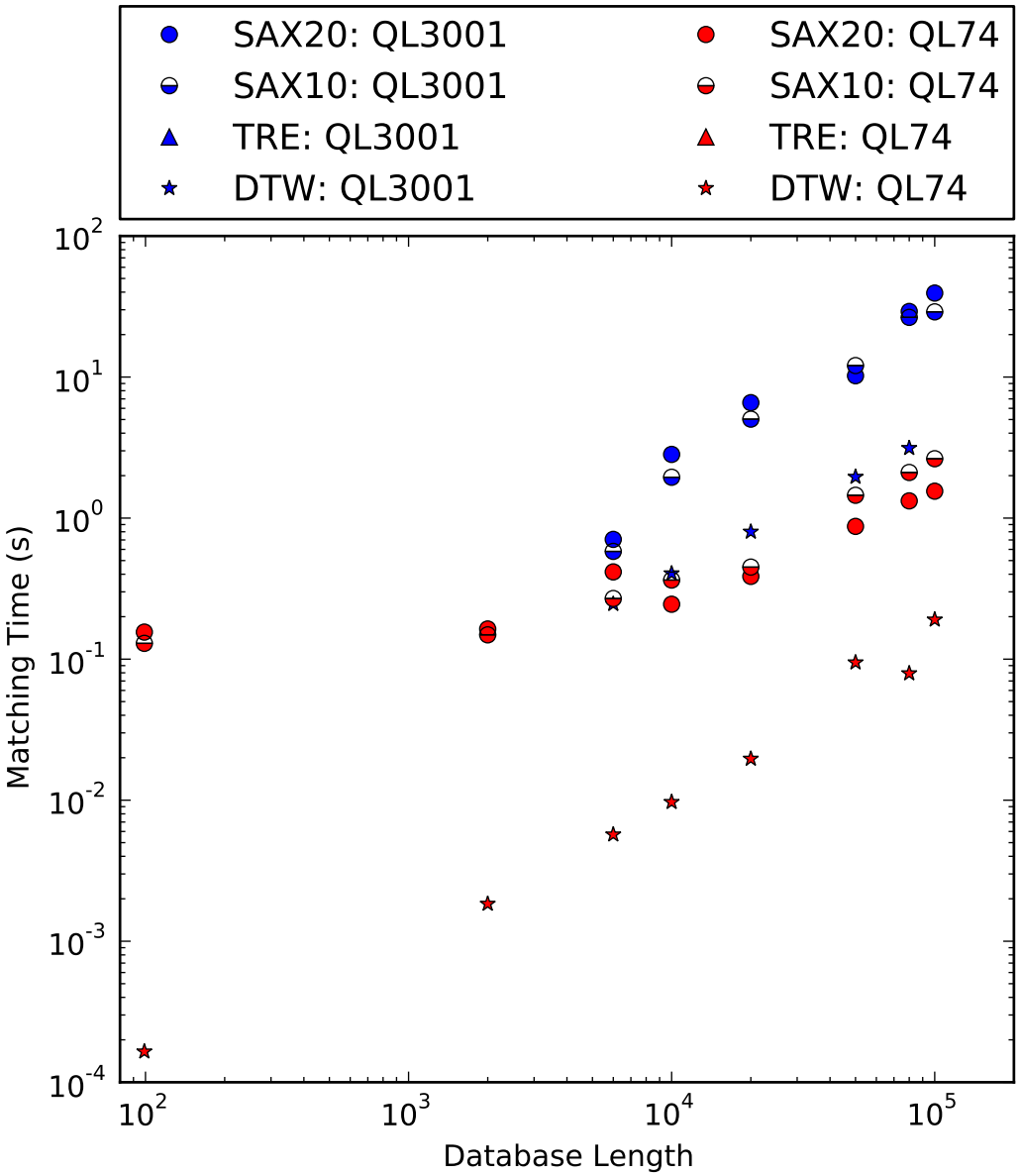

Figure 9.2: Noise-free matching case: DTW is notably faster than the BLAST-based methods in this case, but scales similarly in terms of database size. QL is used as a shorthand to indicate the query length used for the test.

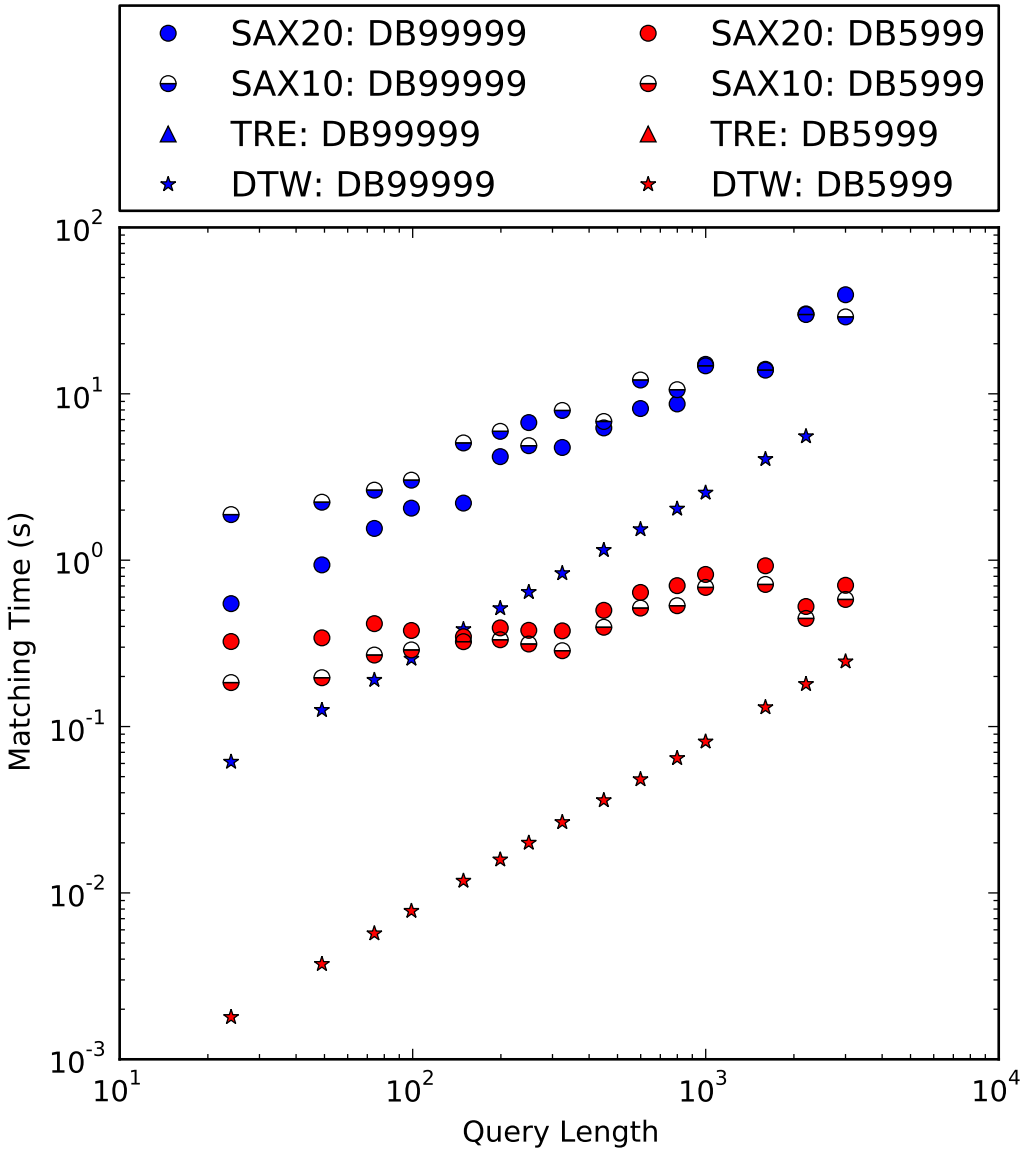

Figure 9.3: Noise-free matching case: It can be observed for a given database length, with no dimensionality reduction, that match time scales linearly for DTW in the best-case situation. DTW is notably faster than the BLAST-based methods for smaller query sizes, but BLAST methods scale better. DB is used here as a shorthand to indicate the database length used for the test.

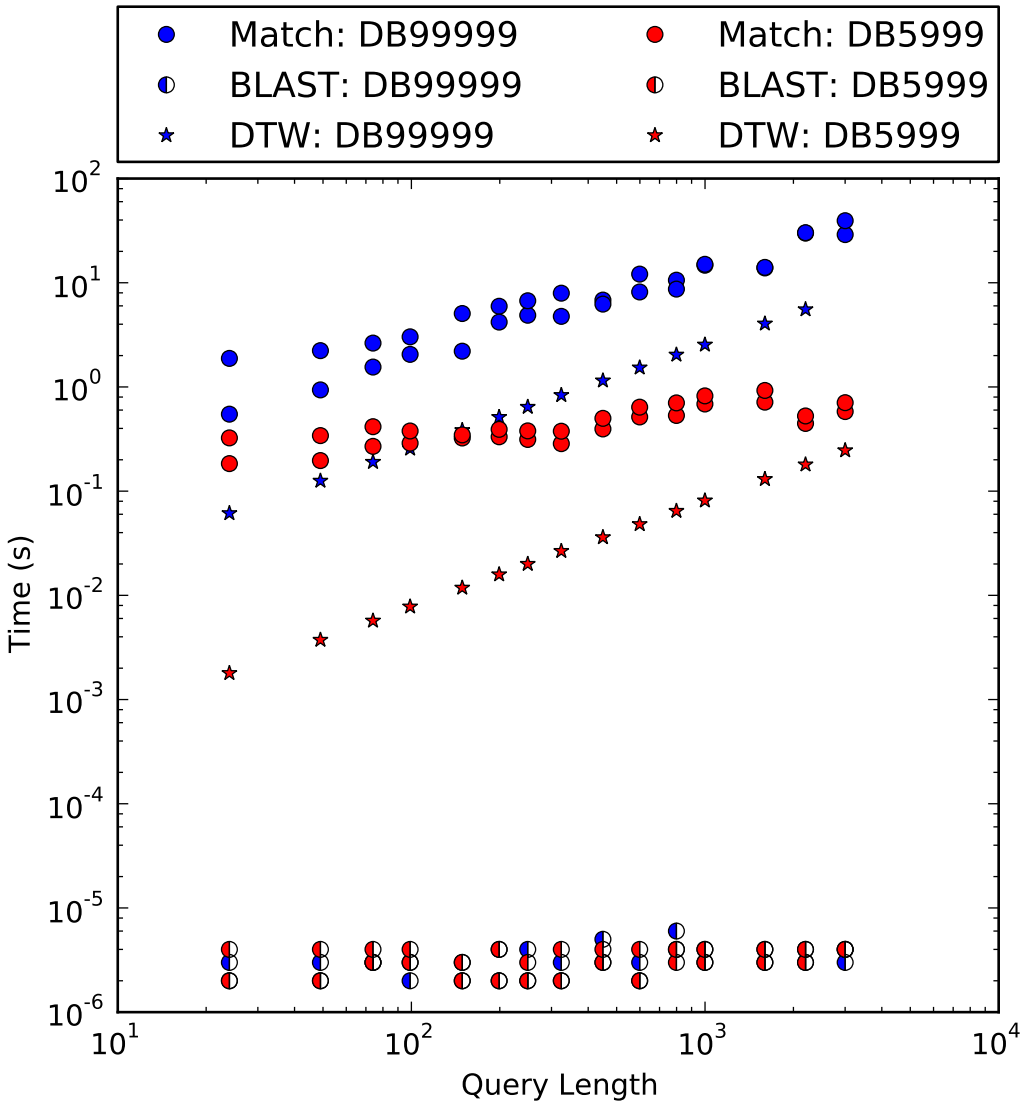

Figure 9.4: The overall matching procedure is far slower than the PSI-BLAST matching takes. This suggests that the implementation of PAA and SAX is the bottleneck in this case. If the SAX-PSIBLAST method is to compete better with DTW, it will be necessary to improve the time taken by SAX in this implementation.

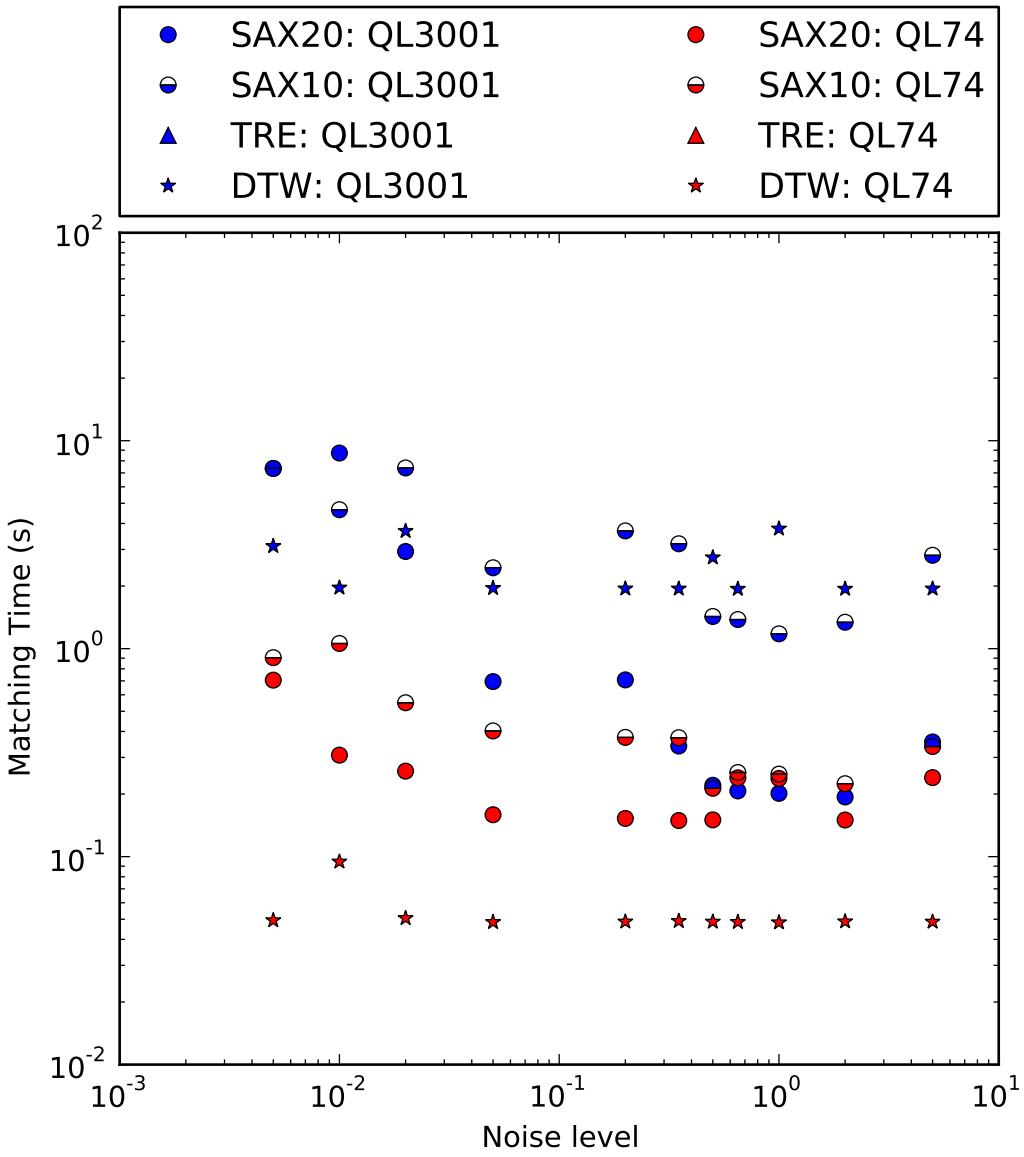

Figure 9.5: Matching time appears to be largely independent of noise level, with the exception of SAX-PSIBLAST, which improves match time as matched seqences get shorter.

9.2.2 Effect of Noise

Figure 9.5 shows how matching speed is primarily unaffected by noise, with the exception of SAX-PSIBLAST. SAX-PSIBLAST appears faster in the noisier cases, but this is due to shorter matches being found in each case, as shown by Figure 9.6.

Figure 9.6 shows that PSIBLAST-based methods will perform faster for a given query size than expected in noisy cases, since the matches found will be shorter. Shorter matches suggest that less gap-extension calculations are required, since in the noisy case the maximum score decrease threshold X will be reached faster. This is because the surrounding data has fewer chance pairings outside the principal match. TER-PSIBLAST does not appear to be as affected as severely as SAX-PSIBLAST, but this is likely due to the filtering which takes place during TER-PSIBLAST. DTW will perform better when

longer matches are found, as it suggests the amount of branching required from a dynamic programming perspective is reduced in these cases.

It is worth noting that when more breakpoints are used by SAX, shorter sequences are returned. This is due to a smaller distance threshold before two time-series values are considered equivalent by SAX. Increasing the number of breakpoints makes SAX-PSIBLAST more selective, at the expense of the length of matches found. Once noise levels exceed the distance between two breakpoints, match quality degrades significantly due to SAX converting the same noise-free values into different characters when the noise is applied. This has a large impact on the scores PSIBLAST will give during gap-extension. For these reasons, the number of breakpoints used can be considered a tuning parameter for SAX-PSIBLAST.

Due to this noise-invariance in matching time, and given the linear relationship shown in Figures 9.2 and 9.3, it can be concluded that the DTW algorithm is $O(mn)$ as stated in literature. It is unclear at this point what the exact order of the SAX-PSIBLAST algorithm is; algorithmic analysis or larger sets of test data are required. However, BLAST-based methods do scale better than DTW with query size.

9.2.3 Realistic process noise

When a white noise level of $\sigma = 0.05$ is used as a simulation of realistic process noise, matching times as in Figures 9.7 and 9.8 are obtained.

The variance in the computational time for PSI-BLAST may be due to the dependence of PSI-BLAST on gap-extension.

Another advantage that SAX-PSIBLAST may offer in terms of computational time is that multiple matches can be performed simultaneously by the PSI-BLAST executable. The testing of this functionality is outside the scope of this study, but may lead to better scaling if multiple simultaneous matches are required.

The naive triangular representation method is hindered by the necessity for filtering before acceptable derivative estimates can be obtained. The filtering step prevents TER-PSIBLAST competing with other methods in terms of computational time. TER-PSIBLAST is still robust to vertical shifts in data, however, and for this reason may still be useful.

9.3 Effects of Dimensionality Reduction

When PAA is used to reduce the dimensionality of data, matching speeds are improved dramatically as Figures 9.9 and 9.10 demonstrate. Although dimensionality reduction can be used for DTW matches, the relative computational load to add this to DTW will be slightly higher. This is because PAA is built-in to the SAX and approximate TR

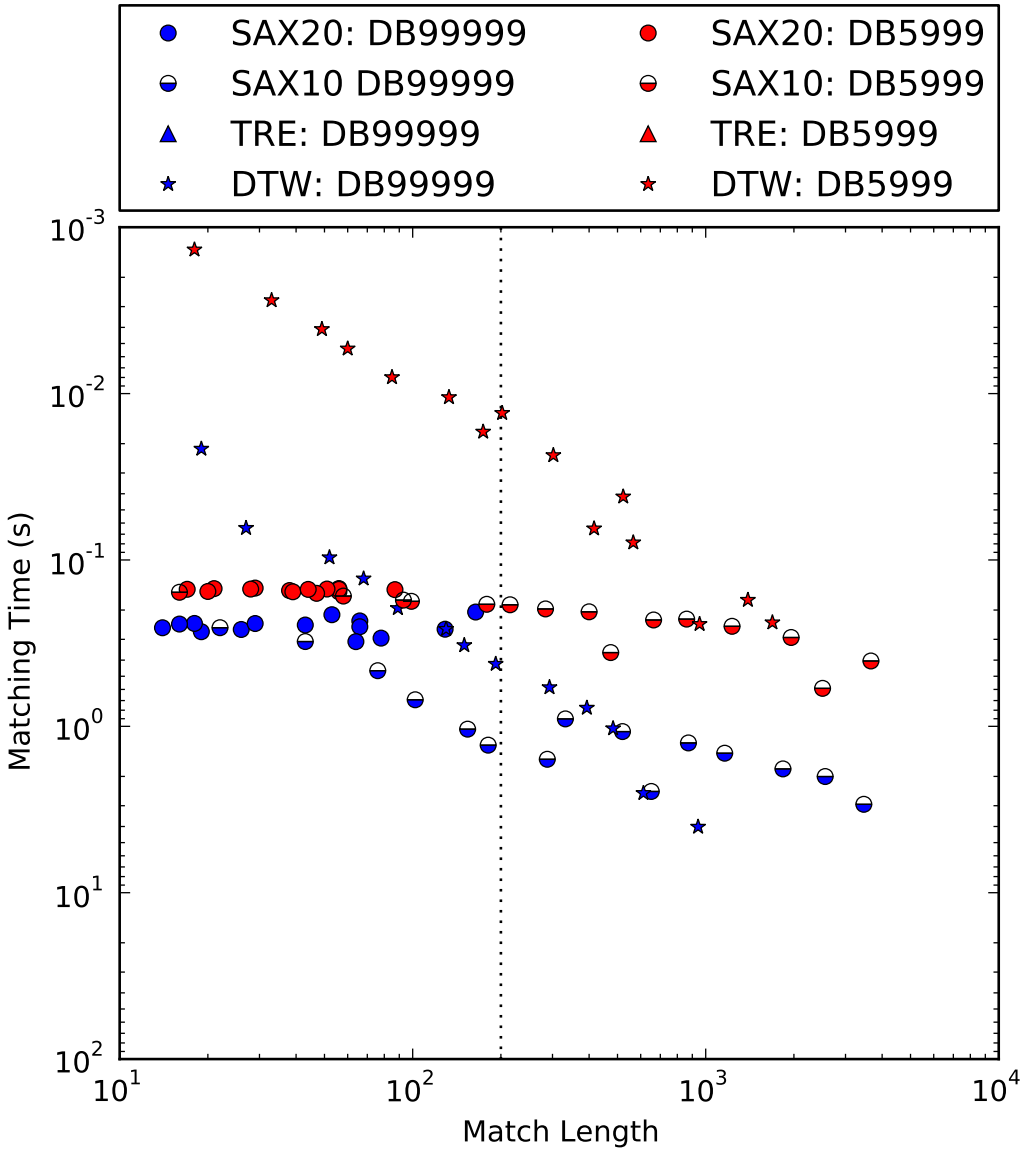

Figure 9.6: In extremely noisy cases ($\sigma = 2$), the matches returned by the SAX-PSIBLAST method with 20 breakpoints are noticably shorter than other methods. This is due to the noiseband exceeding the distance between any two breakpoints, leading to the noisey data being represented by markedly different character strings and reducing PSIBLAST gap-extension scores. It can be seen that SAX-PSIBLAST processing time is also match-length dependent due to gap extension. It can also be noted that as matches get longer, DTW becomes more efficient at finding them, due to pruning of DTW's dynamic programming table.

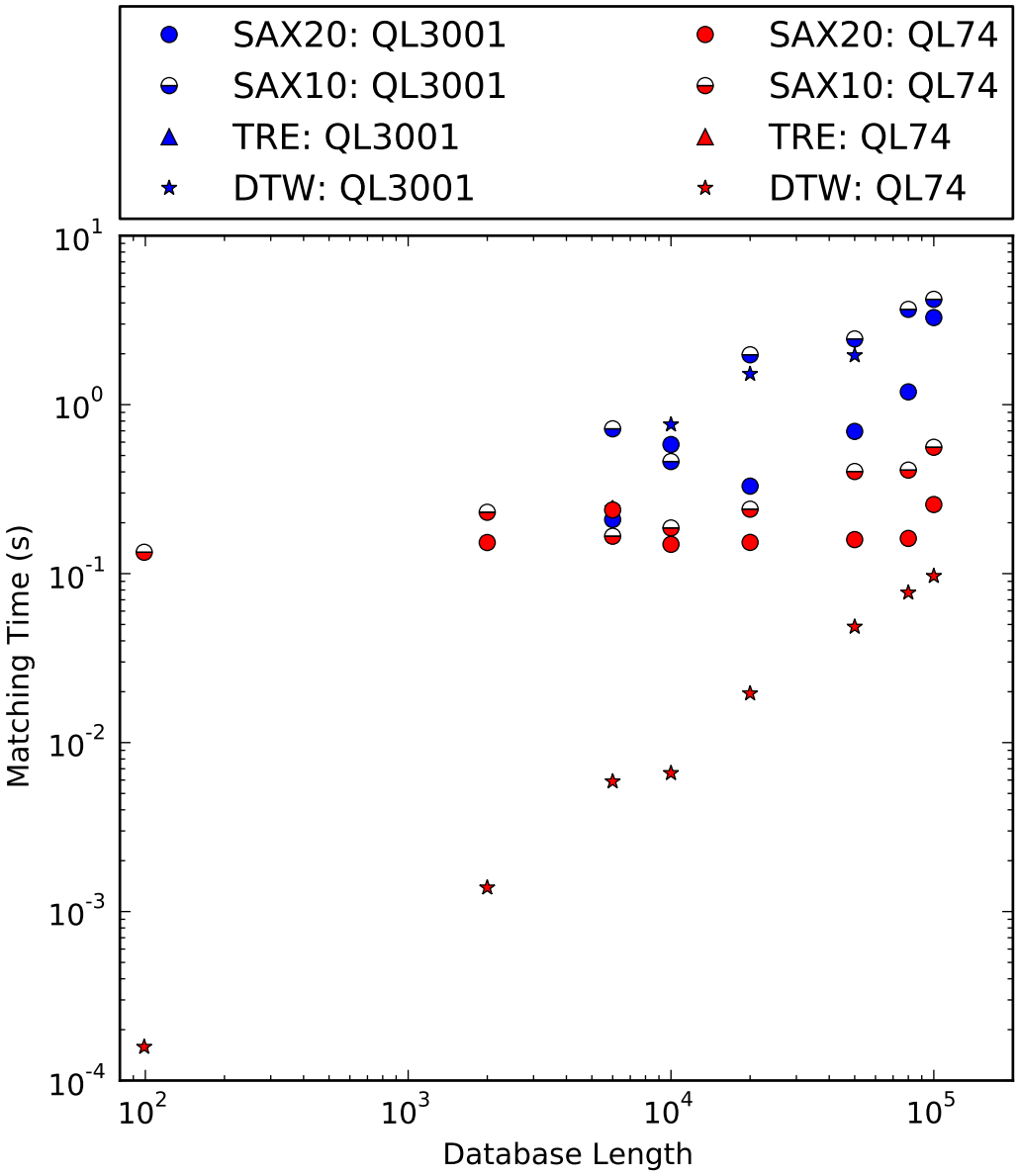

Figure 9.7: Using realistic noise levels, the SAX-PSIBLAST method is competitive with DTW for large databases.

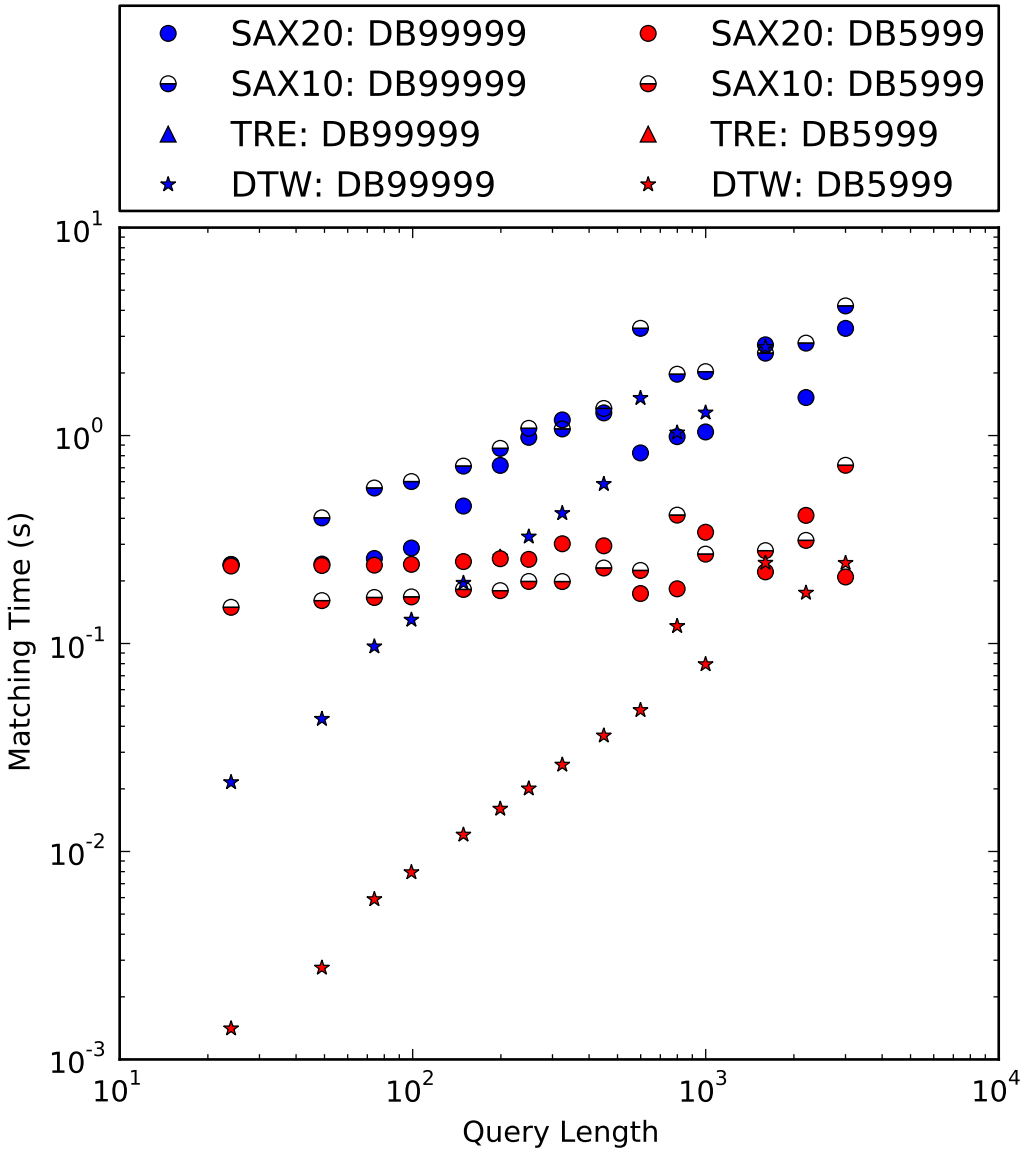

Figure 9.8: Using realistic noise levels, the SAX-PSIBLAST method is competitive with DTW for larger queries. This is also dependent on the database size, but SAX-PSIBLAST scales better for large queries.

procedures in the Python implementation.

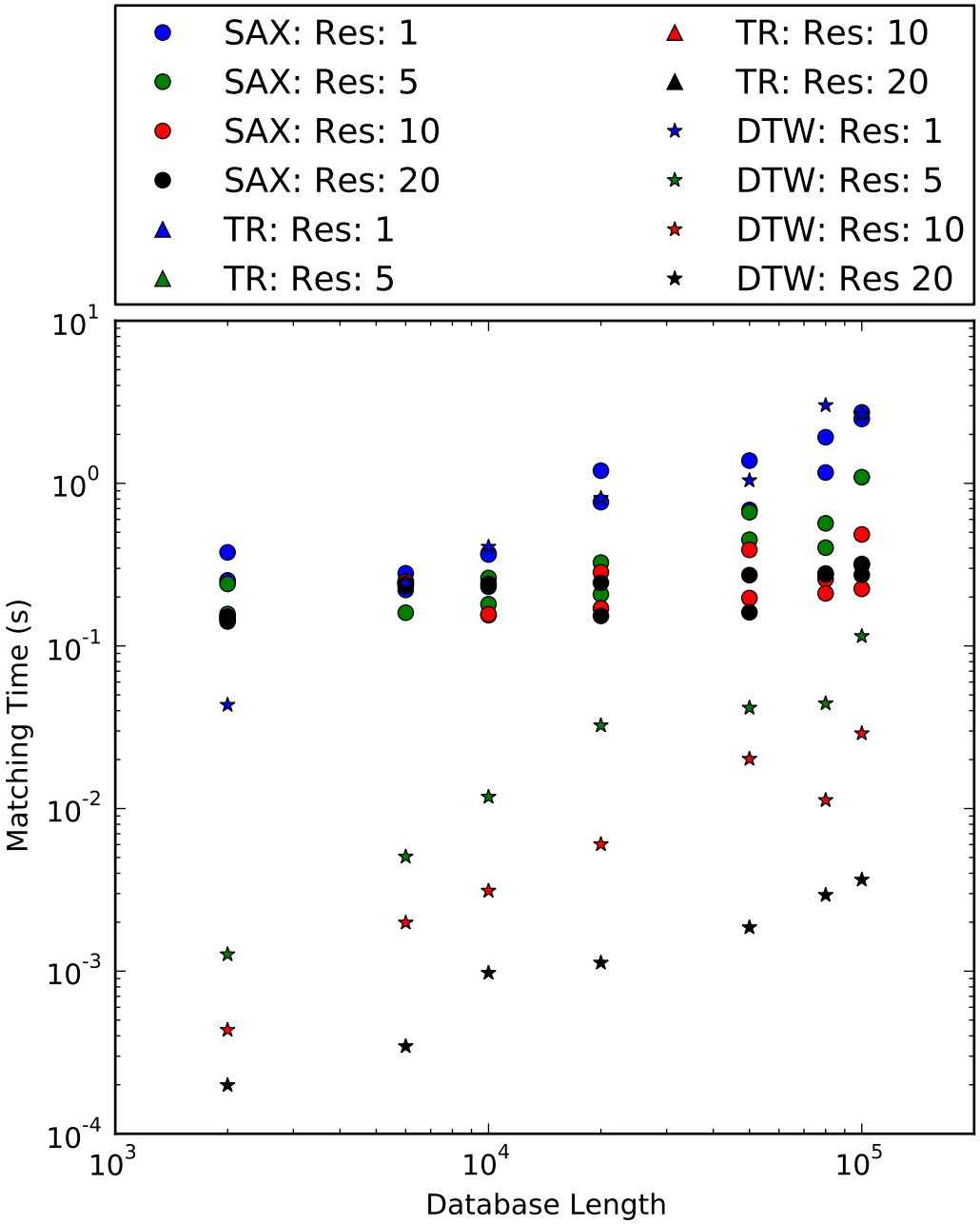

Figure 9.9: When PAA is used to reduce dimensionality, significant gains to matching speed can be expected. A query size of 1599 is used here.

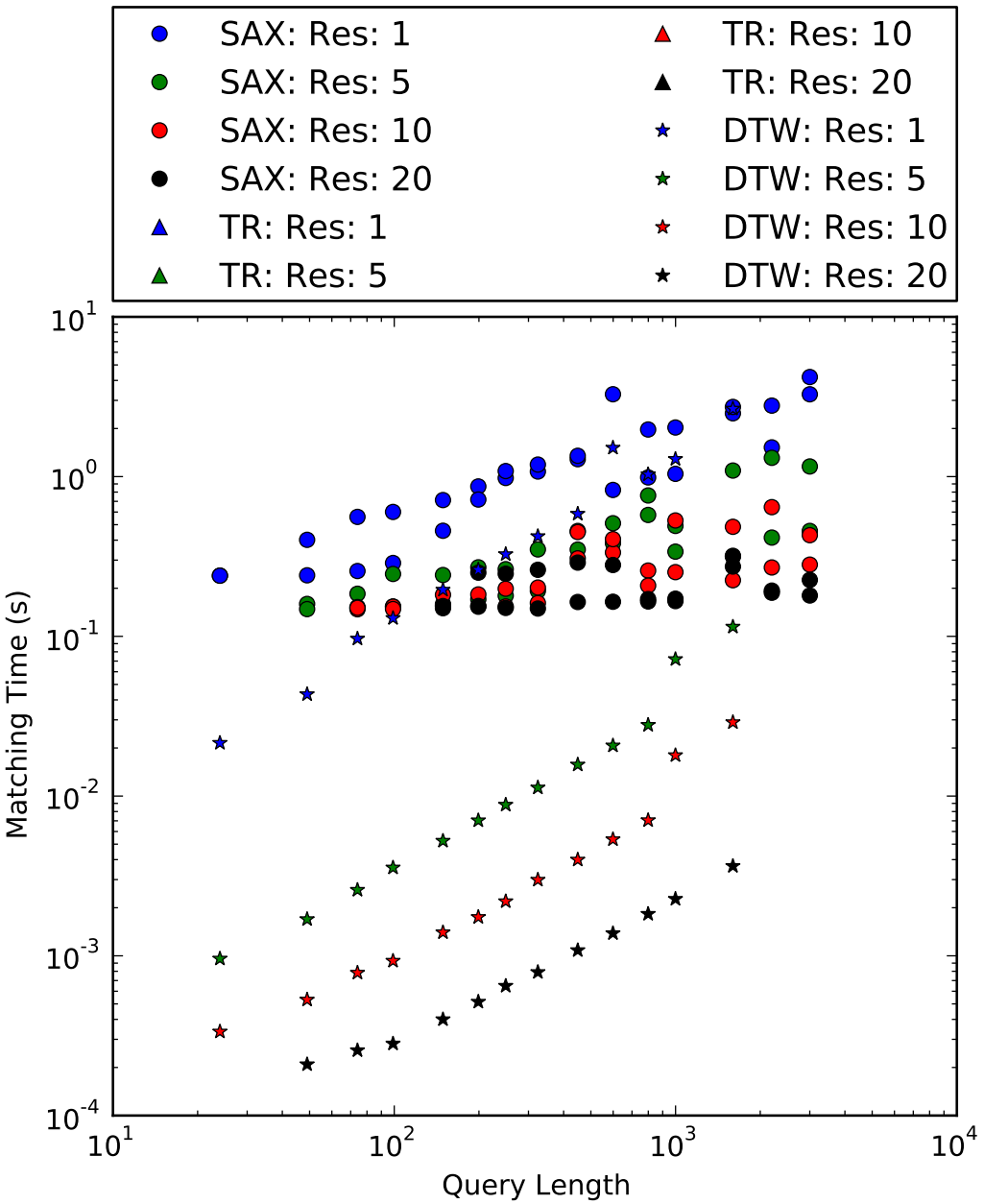

Figure 9.10: When PAA is used to reduce dimensionality, significant gains to matching speed can be expected. A database size of 99999 is used here.

CHAPTER 10

MATCHING ACCURACY

10.1 Comparison of Accuracy Measures

Before the accuracy of matches can be assessed, it is necessary to choose an accuracy measure to use, and whether the accuracy measures tested are independent.

Figure 10.1 shows a comparison between the different scoring metrics for all the matches in the testing procedure. The scores have been normalised by dividing by the match length, as these scores scale with length, then scaled to be between 0 and 1. It can be seen that there is no strong correlation between any scoring metric, implying that they are not interchangeable. This requires that we select SWALE as the scoring metric for match accuracy, for reasons outlined in Section 7.2.2.

10.2 Ability to Find Matches

As Figure 10.2 shows, with no noise, matches are found at the same length as the original query, implying that each method is working correctly. The DTW method has a few anomalous results with no noise, where the match length is not exactly the same as the query. These points refer to the case when dimensionality has been reduced by a factor of five. A possible explaination for this is that shifting of the PAA window will lead to different representations of the same series. Figure 10.3 illustrates how shifting during PAA can influence the dimensionally reduced sequence.

This is the minimum requirement to consider the matching algorithm functional. However, when we increase the noise level of the signal to include some variation, a pattern emerges as in Figure 10.4. DTW consistently returns matches shorter than the initial query, because it is not gap-tolerant, and will return the best sub-match contained within the query. This is undesirable, as potentially valuable process behaviour is discarded.

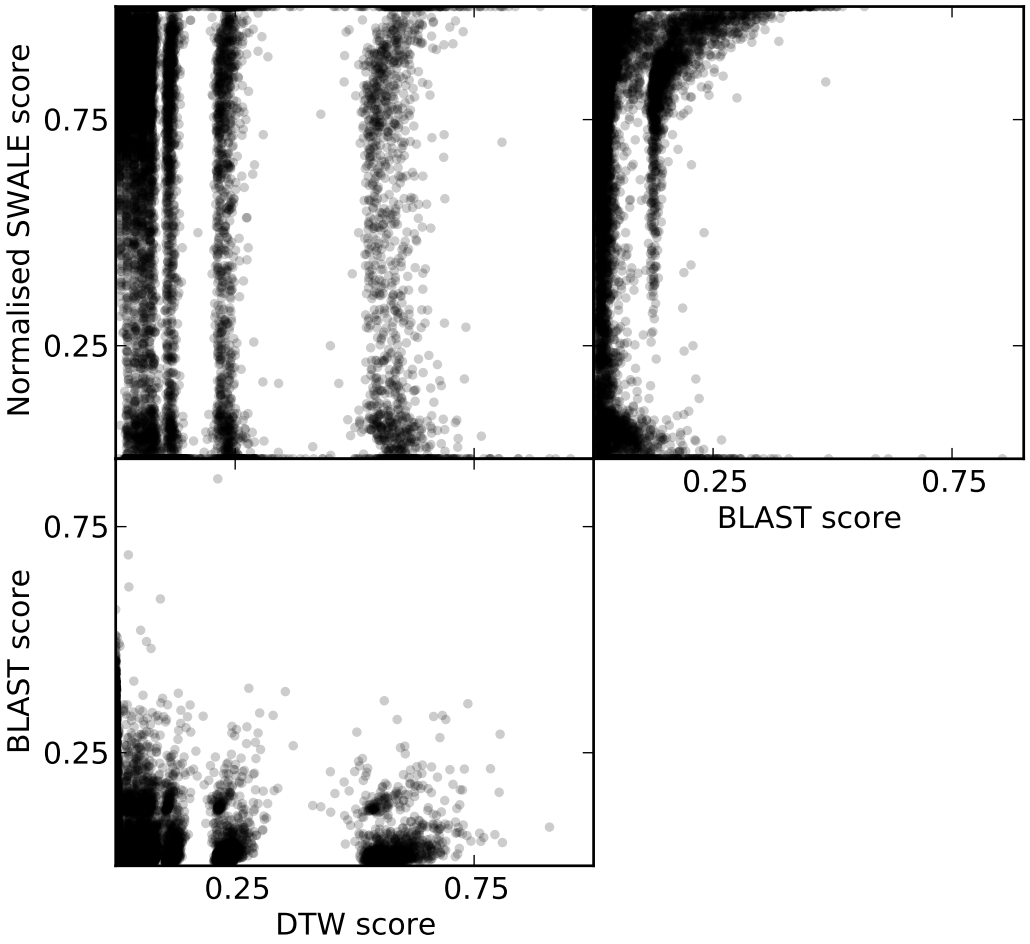

Figure 10.1: It can be seen that there is no strong correlation between SWALE, BLAST and DTW scores for given matches, although it would appear that there is a bounding curve, beyond which no possible combinations exist.

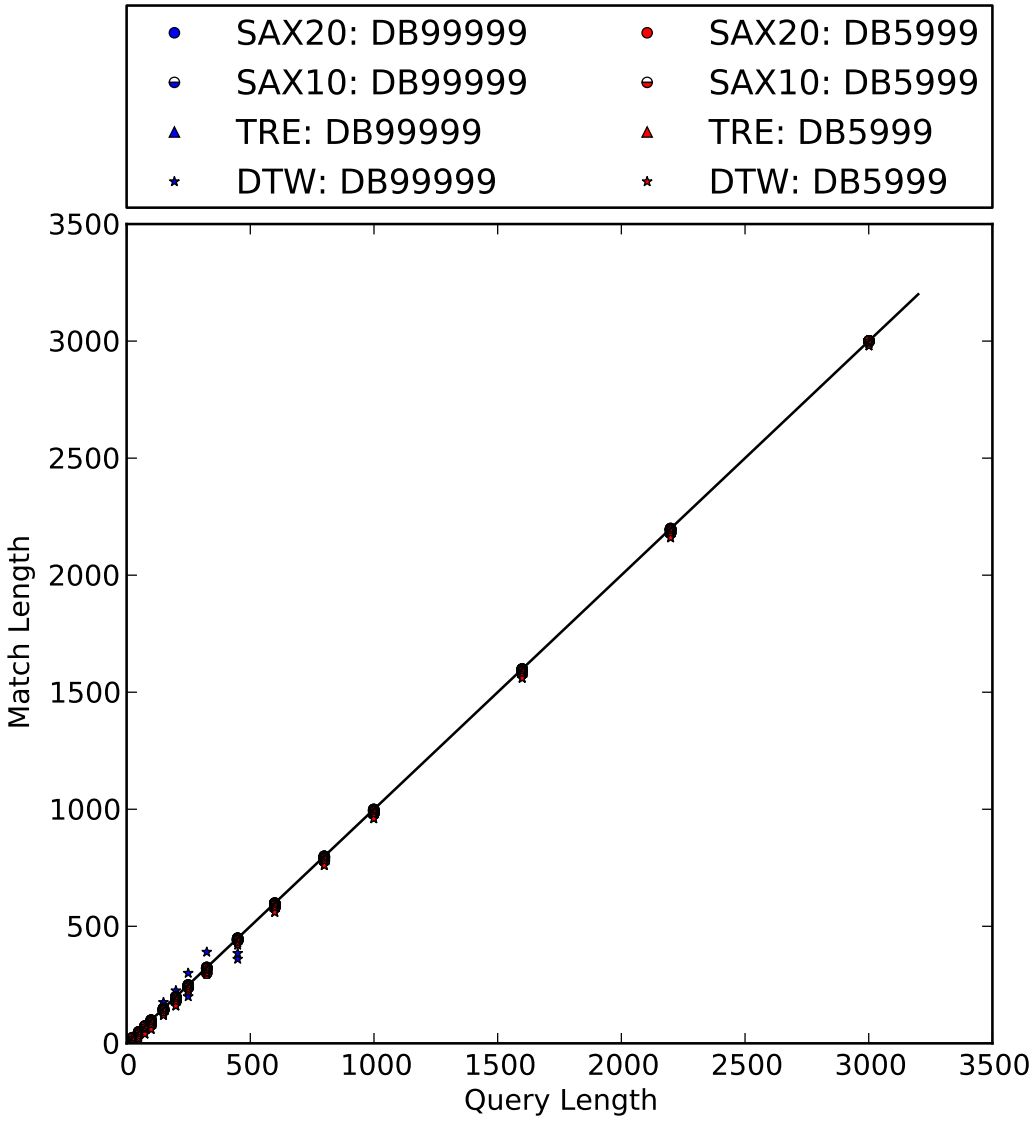

Figure 10.2: The relationship between match and query lengths clearly show that, regardless of dimensionality reduction, a perfect match can be expected by each method when searching for an embedded query. The reduction in lengths by triangular representation matches are due to undefined derivatives at the beginning and end of each query, since one-sided limits cannot accurately predict derivatives.

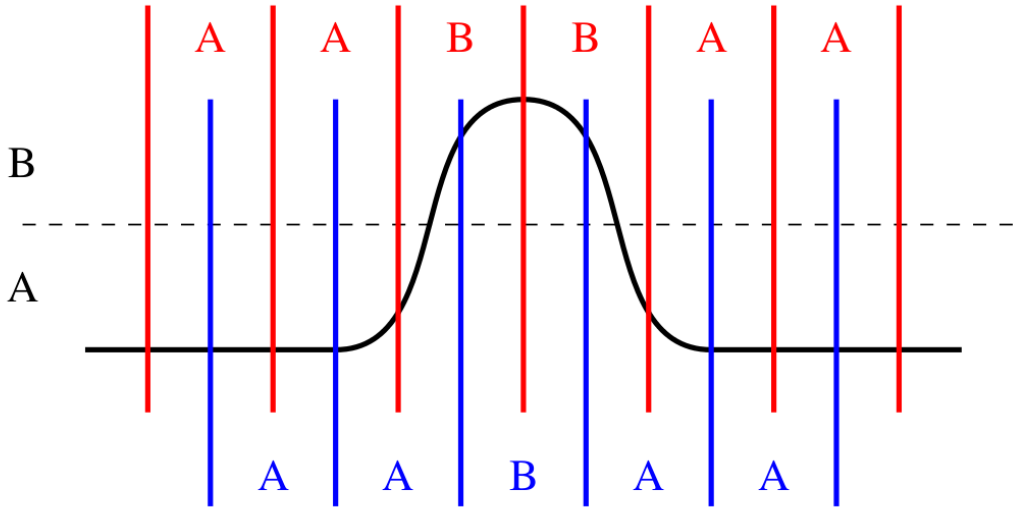

Figure 10.3: An illustration of how shifting a PAA time window can alter the string sequence produced by SAX. The upper sequence output is AABBAA, but the lower sequence is AABAA, even though they represent the same time series. This could affect the results of a DTW match on dimensionally reduced data, even though breakpoints are not used.

PSI-BLAST methods will often return longer matches than queries due to gap extension when believable noise is applied. This is not necessarily detrimental in the case of fuzzy matching, but it must be noted that in test cases this will cause matched sequences to appear erroneous due to adjacent data being included in the match. The tendency of PSI-BLAST to extend matches can possibly be reduced by tuning the scoring matrix to punish gap extension more heavily. Tuning this scoring matrix may require knowledge of the process being modelled, or could be obtained as part of a machine learning exercise. These techniques are not investigated in this dissertation, but will be required in practical applications.

10.3 Match Scores

All methods perform flawlessly with no noise and no dimensionality reduction; perfectly accurate matches are found, as expected.

When considering the hit rates for each method, a normalised SWALE score must be selected as the boundary between what is considered a good or bad match. Two regions are considered in this case, as shown by Figure 10.5.

Because the principal use of triangular representation is for vertically shifted cases, the SWALE score for TER-PSIBLAST is calculated after vertically shifting both the matching subsequences found in the query and database to have the same mean value. This will, unfortunately, inflate TER-PSIBLAST scores when compared to other meth-

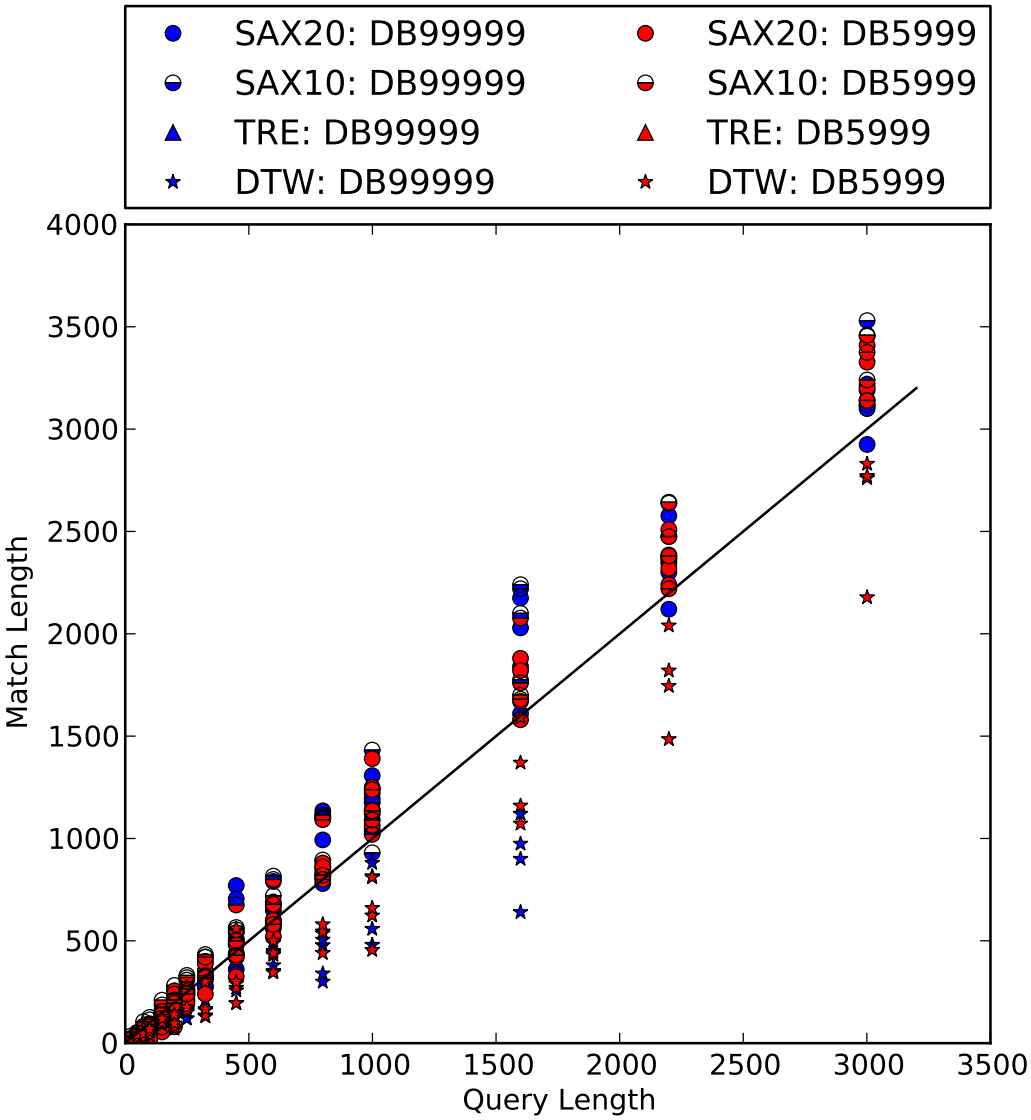

Figure 10.4: BLAST-based methods consistently give longer matches than their initial queries due to gap-extension. This can possibly be avoided by tuning scoring matrices to punish this behaviour more heavily.

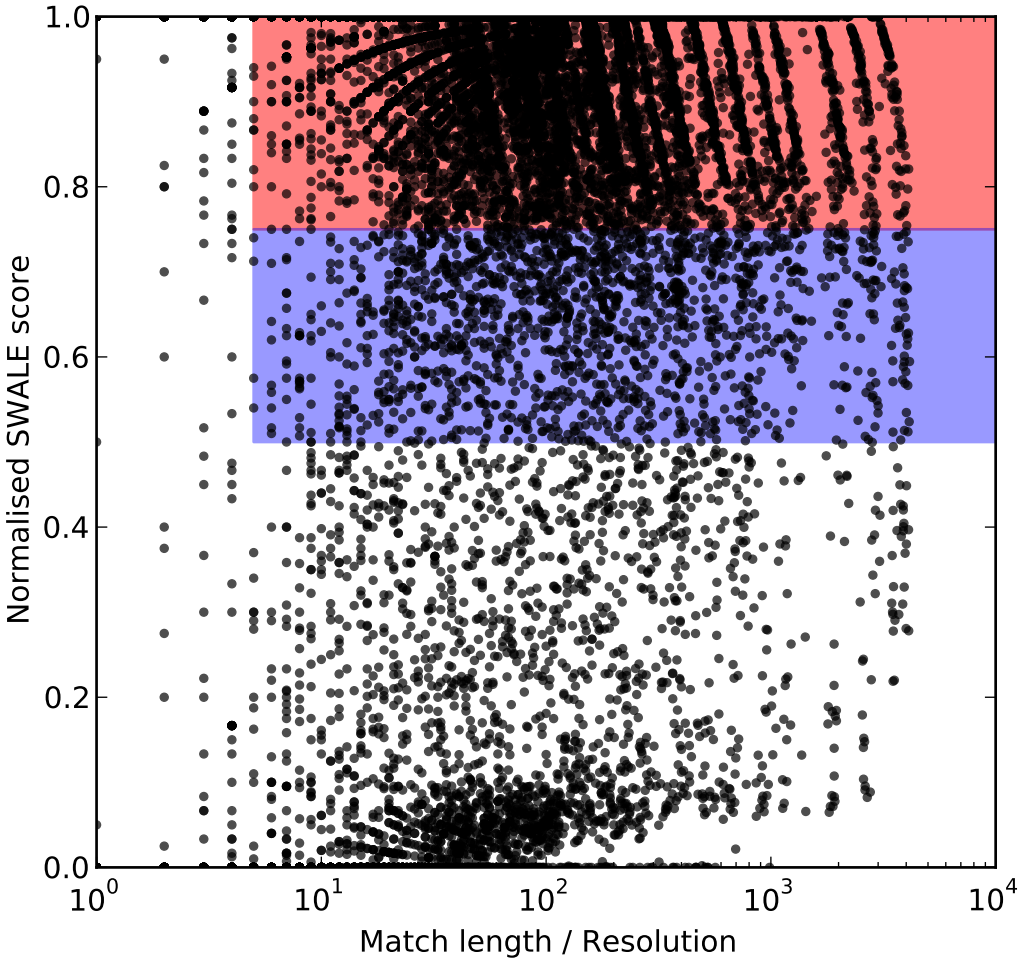

Figure 10.5: Matches are classified as a good match if their SWALE score is larger than 0.5, and an excellent match if they score above 0.75. Only matches with a length 5 times the dimensionality reduction factor are considered.

ods. For this reason, the normalised SWALE scores for TER-PSIBLAST are regarded only for their dependencies on noise and dimensionality reduction, and the results should *not* be used for comparison with other methods.

Figures 10.6 and 10.7 show how the hit rate; the fraction of significant matches with an acceptable SWALE score; degrades with noise level for different levels of dimensionality reduction. This is as expected, but it is interesting to note that the SAX-PSIBLAST method is able to find significantly less matches above a certain noise level. This is likely due to the noise amplitude crossing over the first SAX breakpoint. From this result it can be recommended that the number of breakpoints used be tuned to an appropriate number for a given process noise in practical applications. Due to the filtering effect of PAA, higher degrees of dimensionality reduction improve the acceptable noise levels before SAX-PSIBLAST suffers adverse effects.

Figure 10.8 shows how the *match* rate; the fraction of queries for which a significant match above a given SWALE score is returned; degrades with noise level. SAX-

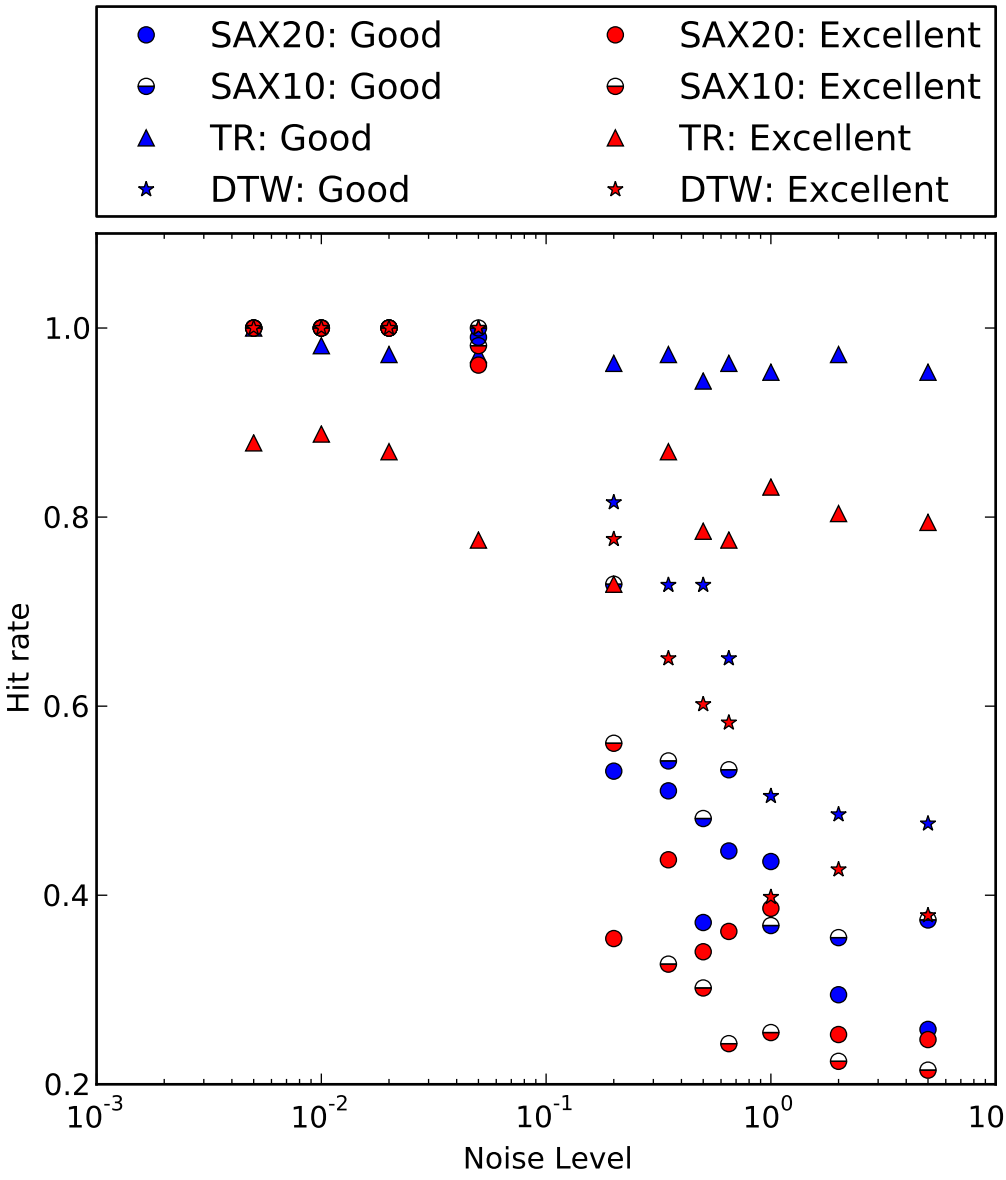

Figure 10.6: With no dimensionality reduction: The hit rate degrades as the process noise levels increase, but SAX-PSIBLAST is affected more severely than DTW above noise levels of 0.1 process units.

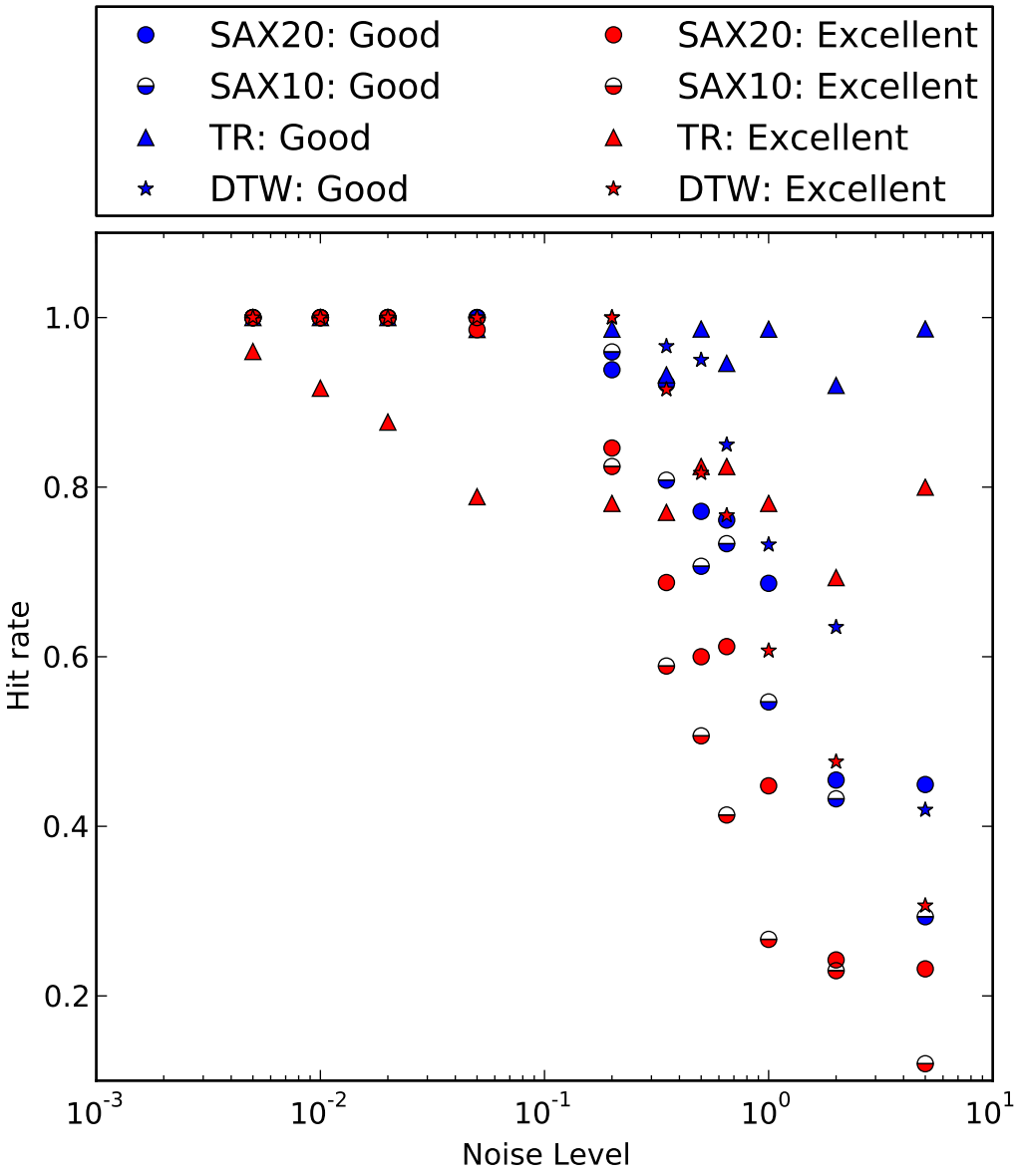

Figure 10.7: With 20-fold dimensionality reduction: The hit rate degrades as the process noise levels increase, but the effect is greater on DTW matches than

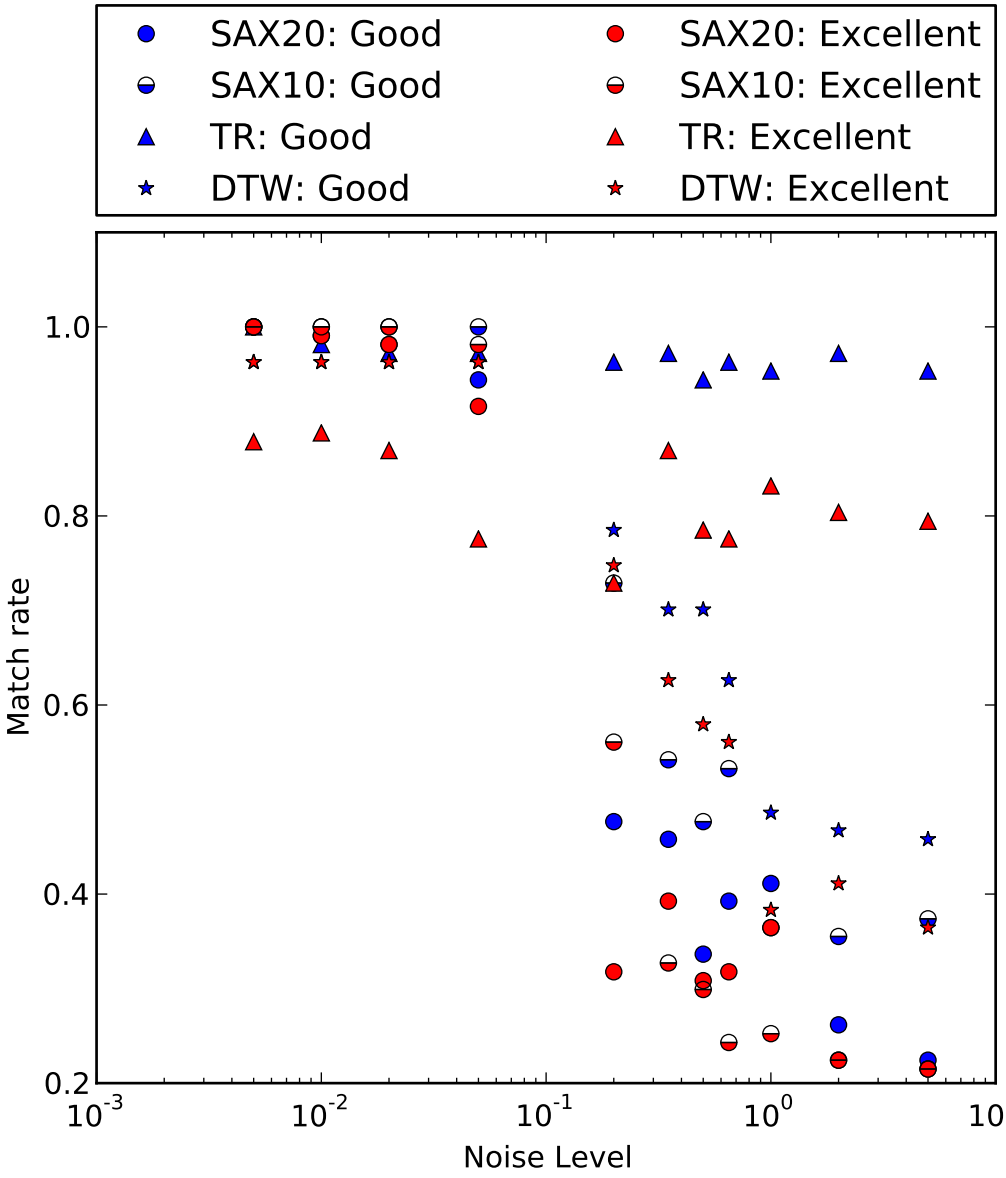

Figure 10.8: With no dimensionality reduction: The match rate of SAX-PSIBLAST degrades rapidly above a noise level of 0.1 process units.

PSIBLAST has a better match rate than DTW for low noise levels, because it is gap-tolerant, but again there is a drop-off in match rates above a noise level of 0.1 process units.

When dimensionality reduction is applied in Figure 10.9, match rates even with marginal noise are not perfect, due to the shifting effect illustrated in Figure 10.3. However, the cutoff noise level for SAX-PSIBLAST has again been improved by the filtering effect of PAA.

Hit rate is also influenced by the level of dimensionality reduction used. Figure 10.10 shows, however, that for the sample process used, a 20-fold reduction in dataset size can be acheived with almost no consequence to the selectivity of the DTW and SAX-PSIBLAST.

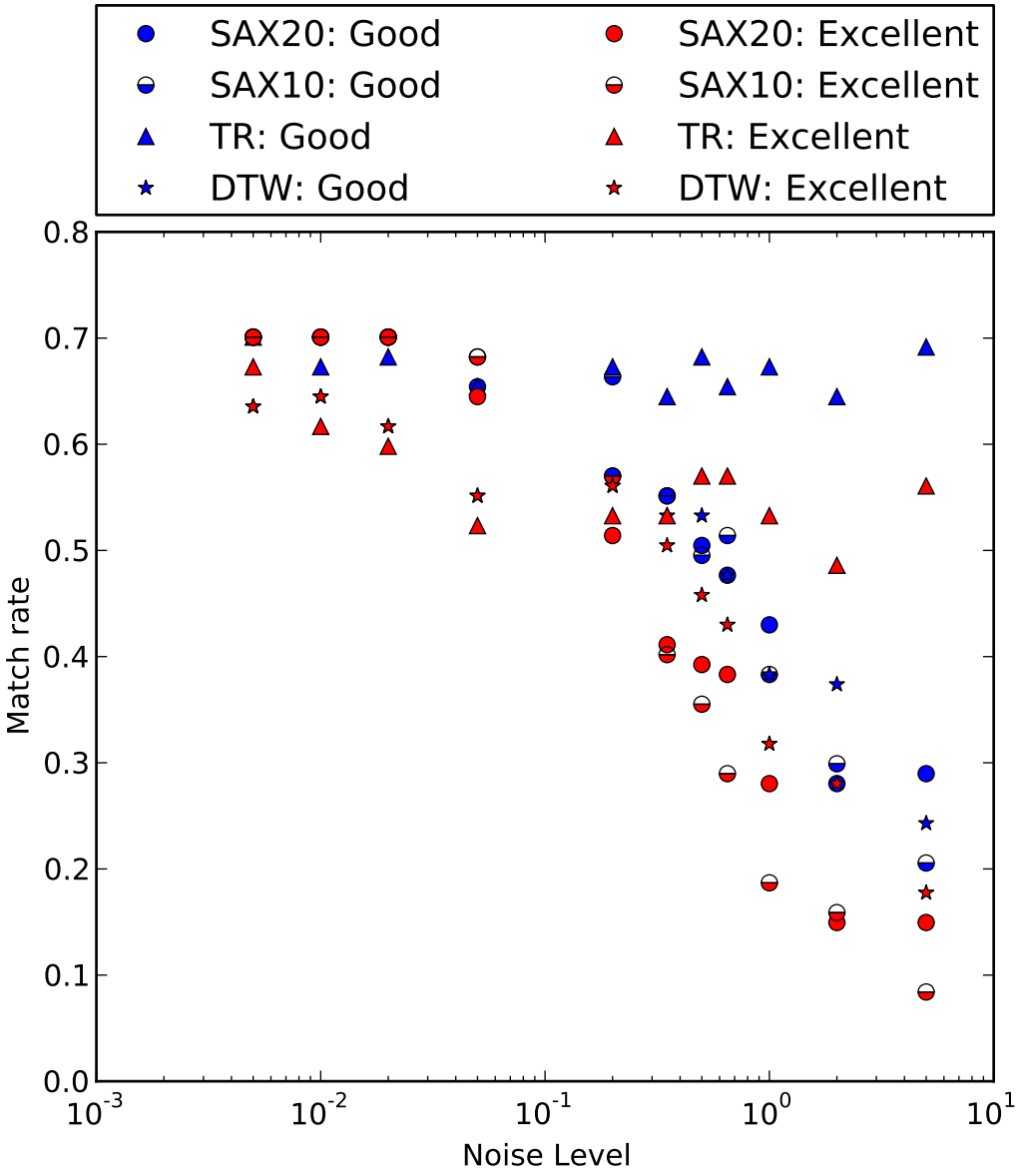

Figure 10.9: With 20-fold dimensionality reduction: Dimensionality reduction causes a lower match rate overall, but the decline in SAX-PSIBLAST power is reduced at higher noise levels due to PAA.

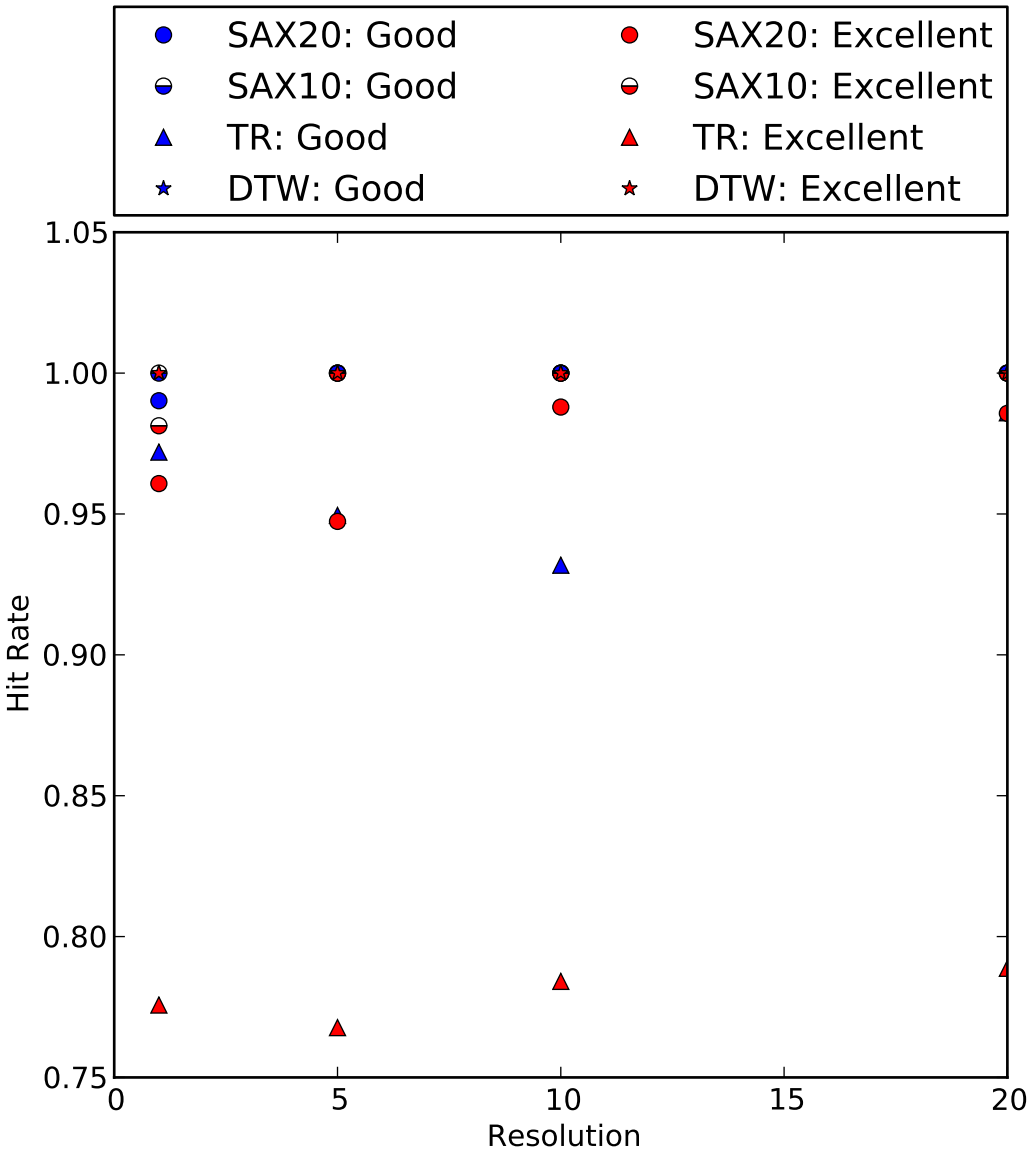

Figure 10.10: With believable process noise, it appears that dimensionality reduction between 1 and 20-fold does not drastically impact the performance of any matching algorithm. The level of dimensionality reduction which will degrade matching accuracy is process dependent, but higher levels of dimensionality reduction than those tested are required to observe this effect on the sample data.

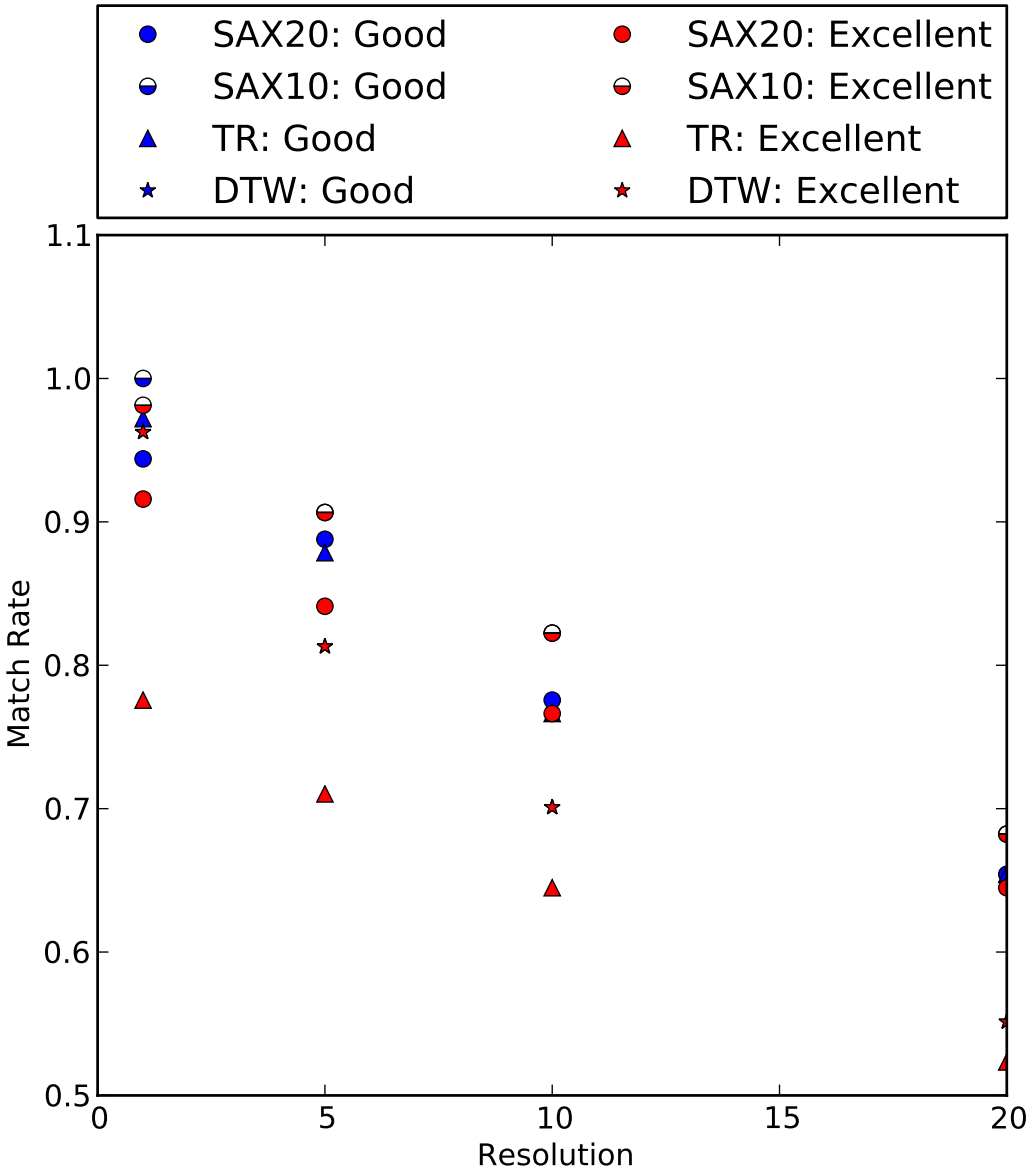

Figure 10.11: With believable process noise, match rates decline with an increase of dimensionality reduction.

Dimensionality reduction has an immediate effect on the number of matches found, as Figure 10.11 shows. This is partially due to shorter queries being reduced to below lengths for which any matches could be considered significant. It is clear that care should be taken when PAA is performed to select a reduction ratio that does not reduce the matching rate to unacceptable levels.

CHAPTER 11

STREAMING PREDICTION FRAMEWORK

The matching approach shown in Figure 7.2 forms part of a larger framework which can be used to match streaming data, and make predictions of future behaviour. By storing the last n datapoints as a query, then following the procedure in Section 7.1, matches to recent data can be generated.

Predictions can be generated by examining the data which occurs *after* the match in the database, showing the process behaviour which followed in each match case. These matches do not require classification of certain process events, nor do they require machine learning. In effect, this prediction model is 'blind'; it does not require process knowledge to perform matches, and will perform matches even on 'uninteresting' data. Classification of motifs and their relationship to process events are not investigated here, but when a classifier is used to limit the time-series database to only pertinent motifs, it is expected that this method will still operate unchanged as part of a larger classifier-matcher framework.

Multiple matching methods can be performed simultaneously for each query, each operating as in Figure 7.2. The methods implemented in this proof of concept are:

- SAX with PSI-BLAST
- Triangular Representation with PSI-BLAST
- Naive Autoregressive modelling
- DTW matching

Figure 11.1 shows a sample output for the proof-of-concept. An execution rate of less than 1 second is feasible for smaller datasets. This is more than acceptable, as industrial processes rarely have dynamic behaviour fast enough to require a faster execution. It

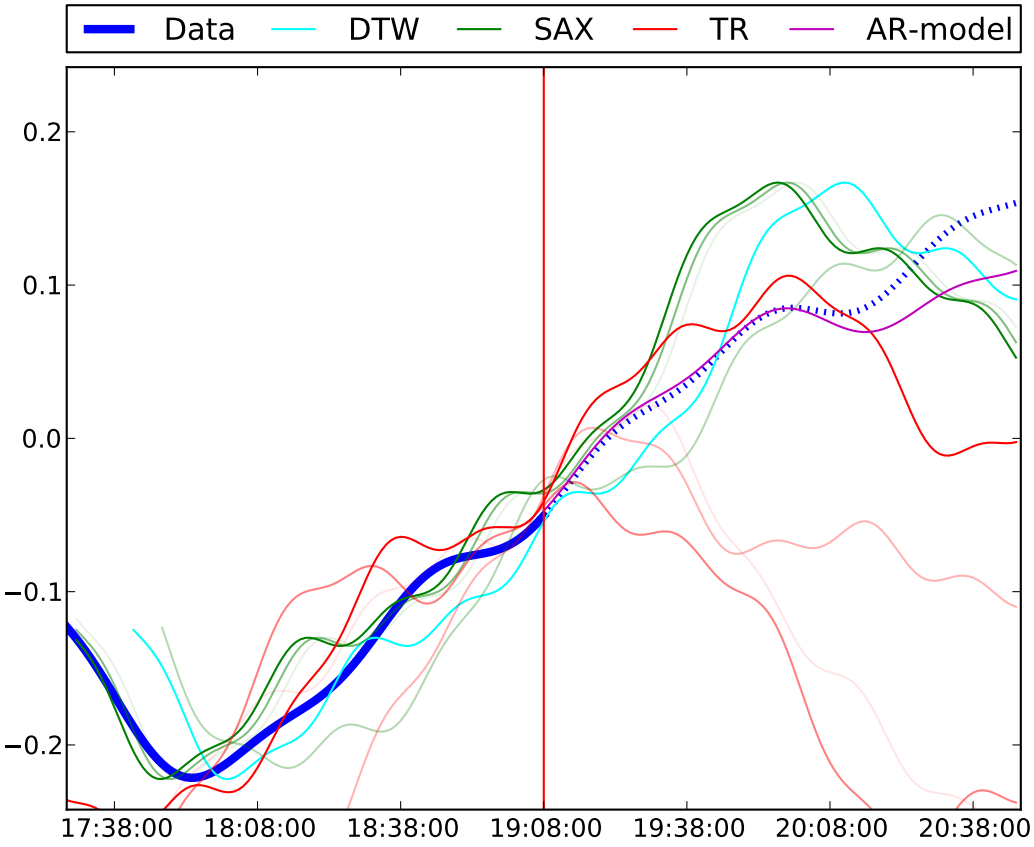

Figure 11.1: Sample output for the proof of concept for online matching and prediction. The vertical line indicates the current time. Execution times of 1s were obtained with a database size of 100 000. Although it would appear that the autoregressive model is performing better than the other methods in use, it is doing so on a non-noisy dataset which was generated using a smoothed random-walk algorithm. This gives VAR an advantage which would not be present in real-world applications. The tuning parameters for each method used are given in Table 11.

should also be noted that processes with faster dynamics will likely require shorter queries, which will allow for faster execution.

Table 11 gives the tuning parameters for each method. No tuning method has yet been developed to obtain optimal results for each algorithm, as it is outside the scope of this dissertation. It is possible that significant gains to the predictive accuracy of each method could be obtained by optimising the tuning for a given process.

11.1 Autoregression

For each query, a new vector autoregressive (VAR) model for the process is obtained using the `vector_ar` functions in the `scikits.statsmodels` library. This model is then used to forecast future behaviour based on the assumptions listed in Section 3.3.

The VAR model does not form part of the initial framework in Figure 7.2, since it is not using historical data to make predictions. The naive implementation of this VAR

Table 11.1: Tuning parameters for real time simulation

Method	Tuning Parameters	Value in Figure 11.1
SAX-PSIBLAST	Buffer size	100
	Breakpoints	10
	Dimensionality reduction	10
	PSIBLAST word size	4
	PSIBLAST e-value	1
TER-PSIBLAST	Buffer size	100
	Boxcar filter size	10
	Dimensionality reduction	10
	Zero tolerance	0.005
	PSIBLAST word size	4
	PSIBLAST e-value	1
DTW	Buffer size	100
	Dimensionality reduction	10
VAR	Buffer size	100
	Dimensionality reduction	10
	Order	67

technique is intended as proof of the concept that additional methods can be added to the modular online matching procedure when required.

The improved prediction of the VAR model is considered to be due to the fact that the sample data used was smooth, without gaussian noise. These data are more easily modelled using VAR than data from real processes due to the pure white noise component used to generate them.

Part IV

Conclusion

CHAPTER 12

CONCLUSION

12.1 Algorithm selection

The SAX-PSIBLAST method handles dimensionality-reduction better, and is more noise-tolerant than Dynamic Time Warping methods for motif-matching, particularly for large queries. Although DTW is faster for small queries, SAX-PSIBLAST is comparable in speed for larger queries, due to it having an order $O < O(mn)$.

SAX-PSIBLAST has the additional advantage of being able to return multiple matches for a given query. SAX-PSIBLAST is capable of finding more matches than DTW, up to a process-specific noise level, beyond which SAX-PSIBLAST is no longer capable of using PAA to reduce the effects of noise.

TER-PSIBLAST is able to perform matches, even with vertical shifts in process data. However, due to the *ad hoc* implementation of the TER-PSIBLAST algorithm, conclusive results on the robustness of such a strategy are not available. Table 12.1 shows how the 3 tested methods compare for different concerns.

Table 12.1: Properties of different motif-matching methods

Property	SAX-PSIBLAST	TER-PSIBLAST	DTW
Hit Rate (Selectivity)	Acceptable	Poor[1]	Good
Vertical Shift tolerance	None	Perfect	Poor
Match speed	Acceptable	Acceptable	Fast
Match speed scaling	$O < O(mn)$	$O < O(mn)$	$O(mn)$
Dimensionality Reduction Tolerance	Good	Poor[1]	Acceptable
Filtering Required	No	Yes	No
Communication Overhead	Yes	Yes	No
Ease of Implementation	Challenging	Easy	Easy

[1]Inconclusive result due to shifting of compared datasets for similarity scoring. Accurate comparisons of these properties to other methods could not be obtained.

12.2 Online Motif-Matching and Prediction

None of the motif-matching methods investigated can be considered dominant, i.e. no method outperforms all others in every field of comparison. It is therefore recommended that any continuous matching procedure should make use of more than one method to perform motif-matching. The method by which multiple results can give consensus on what constitutes a genuine match and alert conditions for process operators are left for future work.

It has been shown that by using the streaming prediction framework, multiple matching methods can be performed online with reasonable execution intervals, and can be implemented without proprietary software. There are several tuning parameters, as shown in Table 11, which could potentially improve the predictive power of the online method.

12.3 Future Work

It may improve the performance of all the methods examined in this dissertation to have optimal tuning. A heuristic tuning procedure, or methods which employ machine learning techniques to optimize the tuning for a given process could be developed for the online prediction case.

Using multiple prediction methods, a weighted voting system could be implemented to flag cases where an undesirable process state is considered imminent. This would benefit process operators, and allow them to take corrective action before a process moves outside of its acceptable operational bounds. There are at the current time, some possibilities for such an arrangement:

- If FTSE is employed to score matches more efficiently than the SWALE method, this would allow for online SWALE-scoring, and could even possibly be part of the weighted voting system for developing a consensus between methods for event flagging.

- A weighted combination of different scoring methods (DTW scores, BLAST scores and SWALE) could be used to find an 'overall' score which could be used as a criterion for reducing spurious notifications to the process operator.

Machine learning elements such as neural networks could be used for online clustering of such events, based on matched motifs, without being used directly for predictions. Combined with the efficient motif-searching capabilities of SAX-PSIBLAST and DTW, such a system would be able to reduce time spent on diagnostics by engineers and operators by returning previous faults from a library of process events.

Future experimental runs can be performed, to determine what effects pre-filtering data will have on matching accuracy and noise tolerance. Efficient filtering may reduce the impact noise has on spurious matching, but may also cause loss of motif information.

It may be possible through better data handling to improve the performance of the Python implementation of SAX. It could also be considered that other programming languages could effect a more efficent implementation than Python.

There is still work required to develop criteria to discern "interesting" motifs from uninteresting ones for fault-finding purposes. This could include the use of PLA to reduce the number of spurious matches, as outlined in Hung & Anh (2007).

BIBLIOGRAPHY

Albanese, D.; Merler, S.; Jurman, G.; Visintainer, R. and Furlanello, C. "Mlpy machine learning py", (2012).

Altschul, S.; Madden, T.; Schffer, A.; Zhang, J.; Zhang, Z.; Miller, W. and Lipman, D. (1997) "Gapped blast and psi-blast, a new generation of protein database search programs", *Nucleic Acids Research, 25(17)*, 3389–3402.

Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W. and Lipman, D. J. (1990) "Basic local alignment search tool", *Journal of Molecular Biology, 215*, 403–410.

Arora, S. and Khot, S. September (2003) "Fitting algebraic curves to noisy data", *J. Comput. Syst. Sci., 67* (2), 325–340 ISSN 0022-0000 URL `http://dx.doi.org/10.1016/S0022-0000(03)00012-6`.

Baum, L. E.; Petrie, T.; Soules, G. and Weiss, N. (1970) "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains", *The Annals of Mathematical Statistics, 41* (1), 164–171.

Berndt, D. J. and Clifford, J. (1994) "Using dynamic time warping to find patterns in time series", Technical report, Stern School Of Business, New York University,.

Bordoli, L. "Similarity searches on sequence databases: Blast, fasta", EMBnet course lecture notes, Swiss Institute For Bioinformatics (2003) URL `www.ch.embnet.org/CoursEMBnet/Basel03/slides/BLAST_FASTA.pdf`.

Brockwell, P. and Davis, R. (1991) *Time Series: Theory and Methods*, Springer Series in Statistics Springer-Verlag, ISBN 9780387974293 URL `http://books.google.co.za/books?id=ZW_ThhYQiXIC`.

Brown, R. G. (1956) "Exponential smoothing for predicting demand", Technical report, Arthur D. Little Incorporated,.

Chan, K. and Fu, A. "Efficient time series matching by wavelets", in *Proceedings of the 15th IEEE International Conference on Data Engineering*, pages 126–133 Sydney, Australia March (1999).

Chen, L.; Özsu, M. T. and Oria, V. "Robust and fast similarity search for moving object trajectories", in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05 pages 491–502 New York, NY, USA (2005) ACM ISBN 1-59593-060-4 URL http://doi.acm.org/10.1145/1066157.1066213.

Cheung, J.-Y. (1992) *Representation and Extraction of Trends from Process Data*, PhD thesis, Massachusetts Institute of Technology, USA.

Cheung, J.-Y. and Stephanopoulos, G. (1990) "Representation of process trends part i. a formal representation framework", *Computers and Chemical Engineering, 14 (4/5)*, 495–510.

Chiu, B.; Keogh, E. and Lonardi, S. "Probabilistic discovery of time series motifs", in *The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, , 1 , 1 pages 493–498 Washington DC, USA August (2003).

Cock, P. J.; Antao, T.; Chang, J. T.; Chapman, B. A.; Cox, C. J.; Dalke, A.; Friedberg, I.; Hamelryck, T.; Kauff, F.; Wilczynski1, B. and de Hoon, M. J. L. (2009) "Biopython: freely available python tools for computational molecular biology and bioinformatics", *Bioinformatics, 25*, 1422–1423.

Dasgupta, D. and Forrest, S. "Novelty detection in time series data using ideas from immunology", in *In Proceedings of The International Conference on Intelligent Systems*, (1995).

Deza, E. and Deza, M. M. (2009) *Encyclopedia of Distances*, Springer, .

Durbin, R.; Eddy, S.; Krogh, A. and Mitchison, G. (1998) *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, ISBN 9780521629713 URL http://books.google.co.za/books?id=R5P2GlJvigQC.

Eddy, S. A. (2004) "What is dynamic programming?", *Nature Biotechnology, 22* (7).

Eddy, S. R. 10 (2011) "Accelerated profile hmm searches", *PLoS Comput Biol, 7* (10), e1002195 URL http://dx.doi.org/10.1371%2Fjournal.pcbi.1002195.

Fonzo, V. D.; Aluffi-Pentini, F. and Parisi, V. (2007) "Hidden markov models in bioinformatics", *Current Bioinformatics, 2*, 49–61.

Fu, T.-c. February (2011) "A review on time series data mining", *Eng. Appl. Artif. Intell., 24* (1), 164–181 ISSN 0952-1976 URL http://dx.doi.org/10.1016/j.engappai.2010.09.007.

Gao, L. and Wang, X. S. "Continually evaluating similarity-based pattern queries on a streaming time series", in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02 pages 370–381 New York, NY, USA (2002) ACM ISBN 1-58113-497-5 URL http://doi.acm.org/10.1145/564691.564734.

Gardner, E. (1975) *Principles of Genetics*, Wiley Sons (Canada), 5 edition.

Han, J.; Dong, G. and Yin, Y. "Efficient mining of partial periodic patterns in time series database", in *Proc. Int. Conf. on Data Engineering*, pages 106–115 (1999).

Haykin, S. (1999) *Neural Networks: A Comprehensive Foundation*, Prentice Hall International Editions Series Prentice Hall, ISBN 9780139083853 URL http://books.google.co.za/books?id=M5abQgAACAAJ.

Hopfield, J. (1982) "Neural networks and physical systems with emergent collective computational abilities", *Proc. Natl Acad. Sci., 79*, 2554–2558.

Hung, N. Q. V. and Anh, D. T. "Combining sax and piecewise linear approximation to improve similarity search on financial time series", in *Proceedings of the 2007 International Symposium on Information Technology Convergence*, ISITC '07 pages 58–62 Washington, DC, USA (2007) IEEE Computer Society ISBN 0-7695-3045-1 URL http://dx.doi.org/10.1109/ISITC.2007.33.

Itakura, F. (1975) "Minimum prediction residual principle applied to speech recognition", *IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-23*, 52–72.

J, E. K. and Pazzani, M. J. (2000) "Scaling up dynamic time warping for datamining", Technical report, University of California,.

Jones, E.; Oliphant, T.; Peterson, P. et al. "SciPy: Open source scientific tools for Python", (2001–) URL http://www.scipy.org/.

Keogh, E. J.; Chakrabarti, K.; Mehrotra, S. and Pazzani, M. "Locally adaptive dimensionality reduction for indexing large time series databases", (2001)a.

Keogh, E. J.; Chakrabarti, K.; Pazzani, M. and Mehrotra, S. (2000) "Dimensionality reduction for fast similarity search in large time series databases", *Journal Of Knowledge And Information Systems, 3*, 263–286.

Keogh, E. J.; Chu, S.; Hart, D. and Pazzani, M. "An online algorithm for segmenting time series", in *In ICDM*, pages 289–296 (2001)b.

Keogh, E. J. and Pazzani, M. J. "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback", (1998).

Keogh, E. J. and Pazzani, M. J. "Pseudo periodic synthetic time series", (1999) URL `http://kdd.ics.uci.edu/databases/synthetic/synthetic.data.gz`.

Kivikunnas, S. "Overview of process trend analysis methods and applications", in *Proceedings of the ERUDIT Workshop on Applications in Pulp and Paper Industry*, University of Oulu, Finland (1998).

Korf, I.; Yandell, M. and Bedell, J. (2003) *BLAST: An Essential Guide to The Basic Local Alignment Search Tool*, O'Reilly Media, ISBN 9780596002992.

Labuschagne, P. J. (2008) "Automatic clustering with application to time dependent fault detection in chemical processes", Masters dissertation, University of Pretoria,.

Levenshtein, V. (1966) "Binary codes capable of correcting deletions, insertions and reversals", *Soviet Physics - Doklady, 10* (8), 707–710.

Lin, J.; Keogh, E. J.; Lonardi, S. and Chiu, B. "A symbolic representation of time series, with implications for streaming algorithms", in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, DMKD '03 pages 2–11 New York, NY, USA (2003) ACM URL `http://doi.acm.org/10.1145/882082.882086`.

Lipman, D. and Pearson, W. (1985) "Rapid and sensitive protein similarity searches", *Science, New Series, 227(4693)*, 1435–1441.

McCulloch, W. S. and Pitts, W. (1988) "A logical calculus of the ideas immanent in nervous activity", in *Neurocomputing: foundations of research*, Anderson, J. A. and Rosenfeld, E. (Eds.), MIT Press, Cambridge, MA, USA ISBN 0-262-01097-6 URL `http://dl.acm.org/citation.cfm?id=65669.104377`.

Molatudi, M.; Molotja, N. and Pouris, A. (2009) "A bibliometric study of bioinformatics research in south africa", *Scientometrics, 81*, 47–59.

Morse, M. D. and Patel, J. M. "An efficient and accurate method for evaluating time series similarity", in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07 pages 569–580 New York, NY, USA (2007) ACM ISBN 978-1-59593-686-8 URL `http://doi.acm.org/10.1145/1247480.1247544`.

National Institute of Standards and Technology "Nist/sematech e-handbook of statistical methods", (2012) URL `http://www.itl.nist.gov/div898/handbook/`.

Pavlidis, T. and Horowitz, S. L. August (1974) "Segmentation of plane curves", *IEEE Trans. Comput., 23* (8), 860–870 ISSN 0018-9340 URL http://dx.doi.org/10.1109/T-C.1974.224041.

Pearson, W. and Lipman, D. (1988) "Improved tools for biological sequence comparison", *Proceedings of the National Academy of Sciences of the United States of America, 85*, 2444–2448.

Pollock, D. (1999) *A Handbook of Time-Series Analysis, Signal Processing and Dynamics*, Number v. 1 in Signal Processing and its Applications Academic, ISBN 9780125609906 URL http://books.google.co.za/books?id=sf5xEq1N8MwC.

Ratanamahatana, C. A. and Keogh, E. J. "Everything you know about dynamic time warping is wrong", in *3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, New York, NY, USA (2004) ACM.

Rath, T. M. and Manmatha, R. "Word image matching using dynamic time warping", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, , 2 , 2 page 521 (2003).

Ross, S. (1996) *Stochastic processes*, Wiley series in probability and statistics: Probability and statistics Wiley, ISBN 9780471120629 URL http://books.google.co.za/books?id=ImUPAQAAMAAJ.

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

Santosh, K. (2010) "Use of dynamic time warping for object shape classification through signature", *Kathmandu University Journal of Science, Engineering and Technology, 6*, 33–49.

Sedgewick, R. and Flajolet, P. (1996) *An introduction to the analysis of algorithms*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA ISBN 0-201-40009-X.

Setubal, J. and Braeuning, R. (2006) *Similarity Search*, National Center for Biotechnology Information (US), .

Smith, S. W. (1999) *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical publishing, 2 edition URL www.DSPguide.com.

Smith, T. and Waterman, M. (1981) "Identification of common molecular subsequences", *Journal of Molecular Biology, 147*, 196–197.

Viterbi, A. (1967) "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Theory, 13* (2), 260–269.

Ye, L. and Keogh, E. J. "Time series shapelets: A new primitive for data mining", in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, DMKD '03 New York, NY, USA (2009) ACM.

Zhang, Y. (2004) "Prediction of financial time series with hidden markov models", Masters dissertation, Simon Fraser University,.

Part V

Appendix

APPENDIX A

SAMPLE MATCH VISUALISATION

Figure A.1 illustrates the visualisation strategy for individual matches. It is performed by the `plotmatch` function. Note that the SAX-PSIBLAST algorithm is gap-tolerant, as seen by the ignored mismatch between datapoints 180 to 350 in the centre graph.

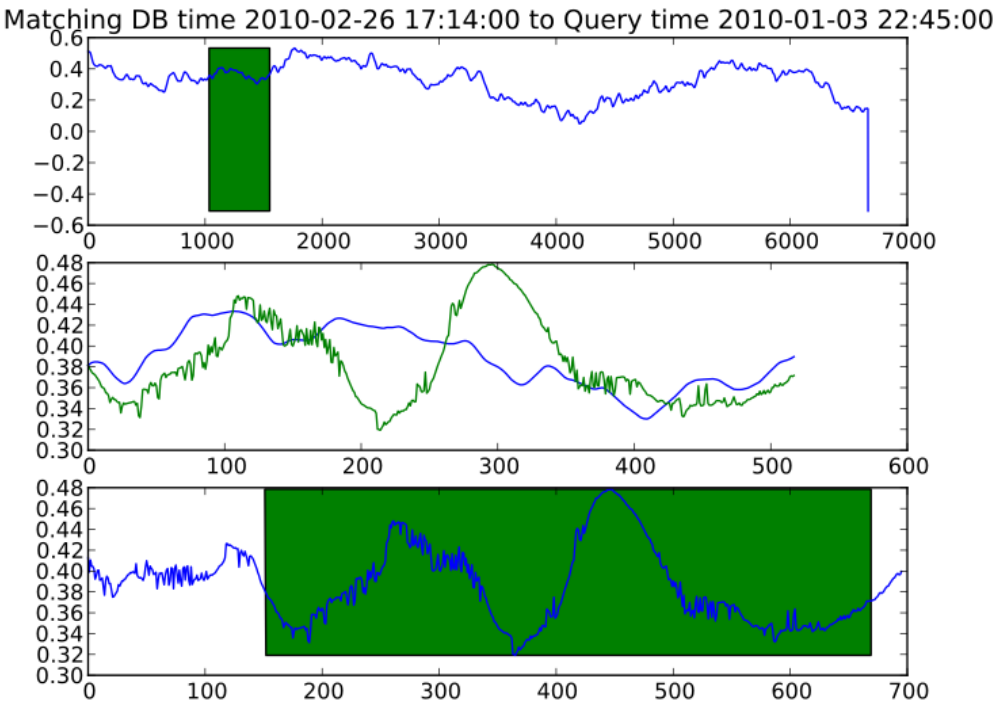

Figure A.1: A sample visualisation for a single match case, generated by the *plotmatch* function. The top graph shows the database segment in which the match was found, the centre graph shows the two matches superimposed, and the bottom graph shows subsection of the query that generated the match.