The Automatic Detection of Patterns in People's Movements

Gordon Forbes

Submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of Master of Science in Engineering.
March 2002

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at this or any other university.

. . . .
Gordon Forbes
(Candidate's Signature)

Acknowledgements

I am grateful to the following for their contribution towards this thesis:

- Professor Gerhard de Jager, my supervisor, for his enthusiasm and guidance.
- DebTech, UCT and the NRF for their financial support.
- The members of the Digital Image Processing group for their support and contributions.
- Liza Burdukova, Brett Hochfeld and Roshan Burtha for their help with the generation of the data sets.
- Keith Forbes for his advice and help with proof reading.

Abstract

This dissertation explores the feasibility of learning the patterns that people walk in everyday scenarios. This information can be used to aid security personnel by drawing their attention to unusual events.

A segmentation algorithm is developed for the fast detection of a marker on a person within an office environment. This is used to create a database of movements.

The DTW (dynamic time warping) algorithm is presented as an efficient method for comparing signals of different length. The DTW is used in conjunction with three clustering algorithms (graph theoretic, Batchelor and Wilkins' and hierarchical) to determine which performs best on 15 data sets with between 14 and 60 movements each. The graph theoretic clustering method correctly classifies 96% of the data, Batchelor and Wilkins algorithm classifies 94% correct and the hierarchical clustering algorithm 73% correct.

The WINEPI (window episode) algorithm is used to generate rules based on sequences of movements that are found in the database. Three tests are run to demonstrate its effectiveness at finding rules in long sequences of events.

Contents

Declaration **i**

Acknowledgements **ii**

Abstract **iii**

1 Introduction **1**

1.1 Overview 3

1.1.1 Data collection 3

1.1.2 Automatic pattern matching 3

1.1.3 Rule generation 3

1.2 Limitations 4

1.3 Literature review 4

2 Data Collection **7**

2.1 Database generation 7

2.2 Segmentation algorithm 8

3 Pre-Processing **11**

3.1 Median filtering 11

3.2 Detecting movement with variance 13

3.3 Further pre-processing 13

4 Dynamic Time Warping **18**

4.1 Distance measures 18

4.2 Linear time normalisation 19

4.3 Dynamic time warping 21

4.4 Constraints 23

4.4.1 Monotonicity constraints 23

4.4.2 Endpoint constraints 23

4.4.3 Local continuity constraints 24

4.4.4 Global path constraints 26

4.4.5 Slope weighting 27

4.5 Dynamic programming 28

4.6 Considerations in dynamic time warping 30

4.6.1 Relaxed endpoint constraints 30

4.6.2 Localised range constraints 31

4.6.3 Multiple time alignments 31

5 Optimising the DTW **32**

5.1 Distance measure 32

5.2 Local continuity constraints 34
5.3 Endpoint constraints 37
5.4 Slope weighting 39
5.5 Global path constraints 39
5.6 False negatives and false positives 40

6 Clustering 42

6.1 Hierarchical clustering 44
6.1.1 Agglomerative clustering 44
6.1.2 Divisive clustering 45
6.1.3 Finding divisions 46
6.2 Graph theoretic methods 47
6.2.1 Similarity matrix 47
6.2.2 Minimal spanning trees 49
6.2.3 Batchelor and Wilkins' algorithm 50

7 Finding The Optimal Clustering Algorithm 53

7.1 Data set generation 53
7.2 Algorithms tested 55
7.3 Comparing data sets 57
7.3.1 Similarity matrix 58
7.3.2 An example 59
7.4 Results 61

7.4.1 Graph theoretic 61

7.4.2 Hierarchical clustering 64

7.4.3 Batchelor and Wilkins 67

7.5 Optimisation 70

7.5.1 Endpoint checking 70

7.5.2 DTW accumulator check 71

7.5.3 A variable distance measure 71

7.6 Conclusions 72

8 Discovering Frequent Episodes 74

8.1 Event sequences 74

8.1.1 Notations 76

8.1.2 Windows 76

8.2 Episodes 77

8.3 Finding frequent episodes 78

8.4 General algorithm 79

8.4.1 Candidate generation 80

8.5 Finding episodes in sequences 81

8.5.1 Serial algorithm 81

8.5.2 Serial algorithm example 81

8.5.3 Parallel algorithm 82

8.5.4 Parallel algorithm example 83

8.6 Complicated episodes 83

9 Finding Episodes in Real Data 85

9.1 Test 1 85

9.2 Test 2 86

9.3 Test 3 88

9.4 Discussion 92

9.4.1 Domain knowledge 92

9.4.2 Serial or parallel episodes 93

9.4.3 Location data or movement data 94

9.4.4 Data sets 94

9.4.5 Event sequence representation 95

9.4.6 Multiple event sequences 95

9.4.7 FPGAs 95

10 Conclusions 96

10.1 Segmentation 96

10.2 Clustering 96

10.3 Finding frequent episodes 97

10.4 Implementation 98

10.5 Further Investigation 98

A Attached CD-ROM 100

A.1 Raw data 100
A.2 Individual movements 100
A.3 Clustering 100
A.4 Event sequences 101
A.5 Serial episodes 101
A.6 Parallel episodes 101
A.7 Serial rules 101
A.8 Test results 101

List of Figures

2.1 A person wearing the pink hat. 8

2.2 Various histograms from the pink hat. 9

2.3 The segmentation process. 10

3.1 The effects of median filtering. 12

3.2 Determining areas of movement. 14

3.3 Merging adjacent movements. 15

3.4 Segmenting movements. 16

3.5 Incorrect segmentation. 16

3.6 Segmentation of a large data set. 17

4.1 Linear time normalisation. 20

4.2 Linear time normalisation not working effectively. 20

4.3 A warping path and its associated warping functions. 22

4.4 Symmetrical and asymmetrical local continuity constraints. 24

4.5 Various local continuity constraints. 25

4.6 Global path constraints I. 26

4.7 Global path constraints II. 27

4.8 Relaxed endpoint constraints. 30

4.9 Allowable warping paths. 31

5.1 Two movements being compared. 33

5.2 Optimal warping paths. 33

5.3 Possible warping paths for various slopes. 34

5.4 Two signals that represent the same movement. 35

5.5 Histogram of signal lengths. 36

5.6 The local continuity constraints that were devised. 36

5.7 Comparison of endpoint constraints. 38

5.8 Various endpoint constraints in use. 38

5.9 Two similar local continuity constraints and their associated path weightings. 40

5.10 Signals with differing degrees of speed of movement. 41

6.1 Data which would best be represented using polar co-ordinates. . . 43

6.2 A taxonomy of clustering approaches. 44

6.3 A typical dendrogram for hierarchical clustering. 45

6.4 A minimal spanning tree and its clusters. 50

6.5 A flow diagram of the modified Batchelor and Wilkins’ algorithm. 51

7.1 Some of the movements walked in the test data sets. 56

7.2 Various error surfaces for graph theoretic methods. 63

7.3 Total error surface for graph theoretic methods. 63

7.4 Various error surfaces for the hierarchical clustering method. . . . 66

7.5 Total error surface for the hierarchical clustering method. 66

7.6 Various error surfaces for Batchelor and Wilkins' maximum distance clustering method. 69

7.7 Total error surface for Batchelor and Wilkins' maximum distance clustering method. 69

7.8 The proposed variable distance measure. 72

8.1 Transforming a data stream to an event sequence. 75

8.2 An alternative method for storing an event sequence. 75

8.3 Graphic representation of various episodes. 77

8.4 Graphic representation of the decomposition of a complex episode. 84

9.1 Start and endpoints for the various movements made in tests 1 and 2. 87

9.2 The four locations used in test 3. 88

List of Tables

7.1 The data sets used for clustering. 55

7.2 Table of parameters for various clustering algorithms. 56

7.3 Results from clustering using graph theoretic methods. 62

7.4 Results from clustering using hierarchical clustering. 65

7.5 Results using Batchelor and Wilkins' clustering method. 68

9.1 Movement labels for test 1. 87

9.2 Locations and their corresponding events. 88

9.3 Results from test 1: frequent serial episodes. 89

9.4 Some of the rules generated from test 1. 89

9.5 Results from test 2: frequent serial episodes. 90

9.6 Some of the rules generated from test 2. 90

9.7 Rules based on die throws for test 3. 91

9.8 Movement start and endpoints for test 3 92

9.9 Results from test 3: frequent serial episodes. 93

9.10 Results from test 3. 93

Chapter 1

Introduction

The automation of security and surveillance is an area of research that has seen a large amount of development recently. The desired product is a *smart* system that will be able to analyse video data in real-time and provide security personnel with additional useful information. This can take on a variety of forms, from signalling when someone enters a room to recording and alerting security personnel to suspicious or unusual behaviour. Most of these systems are not intended to replace the security personnel, but rather to aid them in their task. This is especially useful since in many cases security personnel must analyse many video sequences simultaneously. The chances of them missing an event whilst looking at another one are quite high. This can be greatly reduced by having a smart system that will alert the officers to all unusual events.

The most common smart systems in place at the moment can perform various tasks:

- **Event Detection**
  These systems detect when a specified event takes place. This may be someone entering a room, the number of people in a room reaching a certain threshold or a person walking too quickly.

- **Person Tracking**
  The exact location of people can be tracked as they move around a room or

building. This information can be used to trigger events.

- **Static Object Detection**
  A room can be monitored to see if any new object has been stationary for a long period of time. When this happens, security personnel can be alerted to the presence of the object in the room being monitored. This can be especially useful for the detection of potential bombs.

- **Gesture Recognition**
  Although this is not used extensively in most security systems, certain gestures can be detected. These can be used for a variety of purposes. For instance, a system can be created that turns off the lights in a room when a person waves their arms above their head.

Most of the systems that are implemented have been designed for their specific environment and are therefore not easily portable to different surroundings. New research is being done into systems that will be able to adapt to changing conditions (such as lighting changes). Research is also being done into systems that can be used in many different environments. An example of such a system is the SEDOR (selflearning event detector) system [2] from the company Ascom. This system is able to 'learn' areas in which people tend to move, send alarms when there is movement in other areas and perform static object detection. The SEDOR system goes through a training stage, where it learns the usual behaviour of people. After this it runs continuously.

This work aims to provide an introduction to methods for finding patterns in people's movements. By learning the various movements that people make on regular occasions in an office environment, a smart security system will be able to detect abnormal behaviour and alert security personnel. Apart from the detection of regular movements, the system will be able to analyse the sequences in which these movements occur and make predictions as to what movements are likely to be made next. These predictions can be used to draw the attention of security personnel to events that do not match the predictions.

1.1 Overview

The work presented here is broken up into three main sections:

1.1.1 Data collection

This section deals with the collection of data from an overhead camera. The video stream from the camera needs to be analysed to determine the location of people in the room being analysed. This is a highly complex task in itself. Since this is not the main focus of the work presented here, the segmentation of only one person in the room was performed. Furthermore, to simplify the task and to enable real-time segmentation of the video sequence, a marker was used to identify the person being tracked. Post-processing of the output signal to make the data ready for the pattern matching stage is also discussed.

1.1.2 Automatic pattern matching

The DTW (dynamic time warping) algorithm is introduced as a suitable distance measure to use for comparing signals that have different durations. Various considerations and constraints that can be placed on the DTW are discussed and optimal parameter settings are determined. Three different clustering techniques are compared to find out which will perform the best at determining which movements in various data sets are similar.

1.1.3 Rule generation

Finally the WINEPI (window episode) algorithm of Manilla and Toivonen [20, 21, 28] is introduced as an efficient method for finding rules based on a sequence of events that have taken place over a long period of time. These rules can be used for predicting future movements as well as setting off alarms when unusual movements occur.

1.2 Limitations

This work was limited by the unavailability of a segmentation algorithm with the ability to correctly identify numerous people in a room over long periods of time. For this reason, a simple segmentation algorithm was designed to track one person. Ideally the data source would consist of numerous people's locations in the room. It would then be a trivial task to convert the system discussed here to one that would learn each person's individual patterns. This would result in a system where person A might be allowed to do certain movements, whilst person B doing the same movement would result in the sounding of an alarm.

The unavailability of a very large data set containing video and segmented data over a long period of time was also a disadvantage. This meant that most of the testing had to be performed on relatively small data sets. The result of this is that some of the results (especially those based on the implementation of the WINEPI) algorithm did not perform as well as could be expected on larger data sets. This is simply because that the test data sets were small.

This project is a feasibility study for the various components that would fit into a larger security system. Before a final implementation can be created, a well-defined set of constraints needs to be established. In particular, good domain knowledge is required with respect to the types of movements that one is intending to analyse.

1.3 Literature review

Data mining (also referred to as KDD, knowledge discovery from databases) has emerged as an effective technique for finding association rules, sequences and classifiers [1]. Data mining is frequently described as [17]:

> *The non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from data.*

Implementations of various data mining techniques come in a variety of forms depending on the input data and desired output form. Neural networks have been used to find structure in time series [7]. However, the output from neural networks is generally not in a very usable form and can be difficult to interpret with regard to rule creation in data mining [1]. Chappelier and Grumbach attempted to use a Temporal Kohonen Map to detect the temporal sequences found in signature authentication [4]. Their method of padding with zeros to ensure all sequences are of the same length is not suited to the task at hand.

One of the main issues with data mining is the exceptionally large amount of data that needs to be analysed in many cases. An example given by Keogh and Smyth [16] tells of NASA's shuttles sending back 20 000 readings from various sensors every second. This amounts to approximately 1.7 billion readings a day. Much work is being done on simultaneous compression and data mining for useful information.

In particular, Keogh and Pazzani have worked on using a piecewise linear representation for the data in combination with a DTW algorithm that works on this data representation [15]. They have also done work on making use of suitable indexing schemes to increase the speed of searches in databases [12], an investigation into the automatic weighting of certain features [14] and the use of APCA (adaptive piecewise constant approximation) for the compression of data streams [13].

The current database size used for the tracking of people and the detection of their movements in this project does not yet warrant the increased complexity in implementation of one these aforementioned schemes. Especially since the final implementation will perform a training stage, whereafter the system will be run in real-time. This means that all new data will be analysed and sorted as it arrives, eliminating the need for large database support.

There are numerous clustering techniques that have been developed over the years. Jain *et al.* [11] provide an overview of many of these algorithms. It is important to note that many algorithms may not be suited to a particular problem. In this work, many algorithms that require the desired number of clusters to be specified before the clustering begins, are of no use. This is because this number is unknown.

Various different methods of finding sequential patterns in data sets have been investigated. In particular, four of them that are frequently used are the WINEPI and MINEPI (minimal episodes) algorithm [20, 21, 28, 19], the GSP (generalised sequential patterns) algorithm [27] and the SPADE (special pattern discovery using equivalence classes) algorithm [32, 33, 31].

In a study that compares these four algorithms, Ikizler [9] states that the SPADE algorithm outperforms the GSP algorithm by a factor of two. She also points out that the WINEPI algorithm is more suited to the analysis of telecommunication alarms and gene sequences rather than the analysis of so called basket data (data generated in supermarkets). The data being analysed in this project is more similar to alarm and gene data than it is to basket data.

Chapter 2

Data Collection

In order to detect the various recurring patterns that take place in people's movements, it is necessary to have a large database of location data. This data should be a person's x and y co-ordinates in a plan view of their work environment for long periods of time. The data should also indicate when the person is not present in the work environment.

2.1 Database generation

A large database of movements was not readily available, so it was necessary to generate one. This meant devising an algorithm that could identify a person's location from a stream of video data. This is a very difficult segmentation problem to solve unless various constraints are placed on the work place environment. In order to be able to generate such a database quickly, the following constraints were placed on the work environment:

- Only one camera would be used.
- The lighting in the work place would be controlled so that it stayed at a constant level.

- The subject being tracked would wear a pink hat (marker) to make identification easy.
- Only one person in the work place would be tracked at a time. This would be the person wearing the pink hat.
- Only one pink hat would be present in the environment at any time.
- No pink objects other than the hat were to be in the work place.

A pink hat was used because pink was not present in the work environment. This meant that it would be easier (and therefore quicker to process) to find than a blue or green hat.

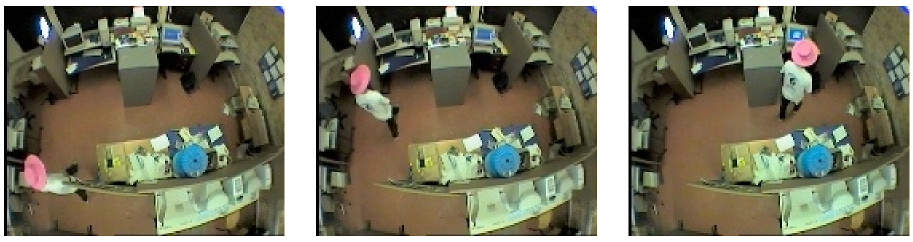

Figure 2.1: Three frames of video showing a person in the work environment wearing the pink hat.

2.2 Segmentation algorithm

The segmentation algorithm was optimised for speed so that it could be run in real-time at a high frame rate (15 frames per second). In order to achieve this, it was necessary to have a simple segmentation algorithm that would run quickly. A certain amount of robustness had to be sacrificed in order to achieve this. These sacrifices are in effect the constraints mentioned above.

The algorithm for segmentation runs through the following steps for every frame in the video sequence. It does not make use of any information from the previous frame(s).

1. Load the current frame.
2. Detect all pink areas in the image.
3. Label all of the pink areas.
4. Determine which of the pink areas is the largest.
5. Determine the centroid of this area.

The pink areas in the image were detected by placing various thresholds on the R, G, B (red, green, blue) values of the pixels in the image. It was noted that the colour of the hat in terms of R, G, B values differed when the person was in various locations in the work place. In particular, the centre of the work place and the edges differed greatly. This is shown in figure 2.2

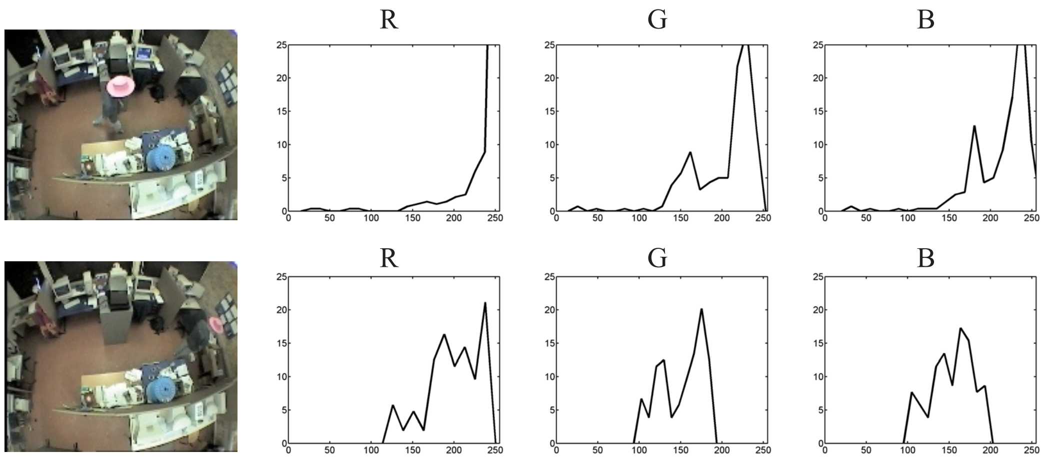

Figure 2.2: TOP: Histograms of R, G, B values of the pink hat when the subject is at the centre of the work place; BOTTOM: Histograms of R, G, B values of the pink hat when the subject is at the edge of the work place.

Based on these histograms and further analysis of the colour pink, four decision criteria were determined to distinguish the pink from other colours:

- R,G,B values must all be greater than 90.

- The value of R - B must be greater than 15.
- The value of G - B must be less than 20.
- The values of (R, G, B) must not be in the ranges (120±30,93±30,63±30).

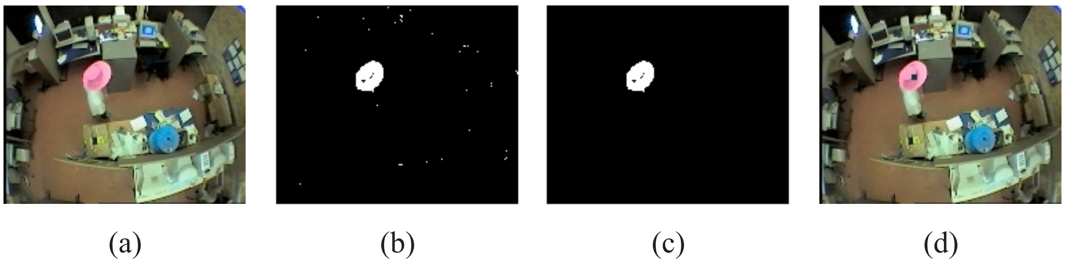

Figure 2.3: (a) Original frame; (b) pink regions; (c) largest pink region; (d) centroid (indicated by the black marker).

The centroid of the largest pink region (the hat) is used to determine the person's *x* and *y* co-ordinates. Since the hat will normally appear as either an ellipse or a circle, the centroid will give a good indication of its centre.

The centroid is defined as:

$$C = (C_x, C_y) \tag{2.1}$$

where

$$C_x = \frac{\sum_i x_i}{Area} \qquad C_y = \frac{\sum_i y_i}{Area} \tag{2.2}$$

and *Area* is the number of pixels present.

Chapter 3

Pre-Processing

Once the database of *x* and *y* co-ordinates is available, it is necessary to convert it to a database that contains the individual movements of the person being tracked. In other words, it is necessary to remove all of the data when the person was not in the work environment as well as when the person was remaining stationary for any significant period of time. Once this has been done, all that will remain is the individual movements that the person made. These movements can then be further analysed so as to 'learn' the patterns of movement that have been made.

3.1 Median filtering

As mentioned previously, the segmentation algorithm that detects the person's hat (and therefore their location) is not very robust. This means that it might incorrectly indicate that the person being tracked is in the room when they are not. This normally only happens for a very short period of time. Because of this, filtering can be used to eradicate such spurious outputs from the segmentation algorithm.

A median filter was chosen instead of various other filter types (mean, gaussian) as it is very good at removing salt and pepper noise. This is non-gaussian noise that has extreme values and is very similar to the noise shown in figure 3.1. The

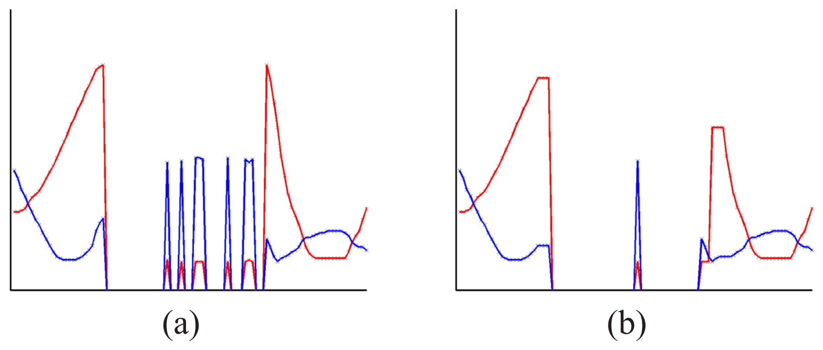

Figure 3.1: The effects of median filtering. (a) The *x* and *y* co-ordinates of a person being tracked. Five erroneous "spikes" are present, caused by errors in the segmentation algorithm. (b) The *x* and *y* co-ordinates after the "spikes" have been removed by median filtering.

median filter also has an edge-preserving property which is very important. There are many sharp edges in the data and it is important that they are maintained when filtering is performed. The edges are formed when the subject being tracked leaves the work environment. If the filtering process did not preserve the edges, it would appear as if the person had 'teleported' a large distance.

A seven point running median filter was applied to the data stream. As can be seen in figure 3.1 this results in the minimisation of spurious data. One of the unwanted side effects of the filtering process is the rounding off of some of the sharp edges in the data. This is not a problem since the rounding off of the edges should happen in a similar manner to all of the edges in the data. Also, this will be accounted for when the dynamic time warping function is used with relaxed endpoint constraints (see chapter 4). Not all of the spurious data is removed from the data (figure 3.1). This is not a major problem, as it is more important to remove the spurious data that is present when the subject is in the work environment, than it is to remove spurious data when there is no one present. Furthermore, if there is too much spurious data present, the movement will not be selected from the data stream for further processing.

3.2 Detecting movement with variance

When the person being tracked is standing still or is out of the work environment, his x and y co-ordinates will remain constant for a certain period of time. The variance of a set of constant data will always be zero. This fact can be used to determine whether or not motion is taking place in the work environment.

The sample variance unbiased estimator (s^2) for σ^2 is used in this work.

$$s^2 = \frac{1}{N-1}\sum_{i}^{N}(x_i - \overline{x})^2 \tag{3.1}$$

For each data point in the data stream, the variance of the data point and the ten data points on either side of it is calculated. This gives an indication of whether or not the data point is part of a movement.

A threshold is then applied to both the x and y variance streams. This is used as the basis of whether or not movement is happening. Once this has been done, the two streams are merged, by taking the maximum value of the two streams. This is the same as the logical OR command. This is done because a person can walk around the office environment so that one of his co-ordinates remains constant while the other one varies.

3.3 Further pre-processing

After applying a threshold to the variance, further pre-processing needs to be done. The first stage is to search through all of the areas of movement in the data stream and to determine if any adjacent movements should be combined into one movement. This is done by determining the size of the gap between the two movements, and merging them together if the gap is considered small. This means that if the person being tracked stops very briefly and then continues walking, it will be considered to be one movement.

Whenever the person being tracked leaves the work environment, there is a sharp edge in the data stream. This means that the there will be a large variance in the

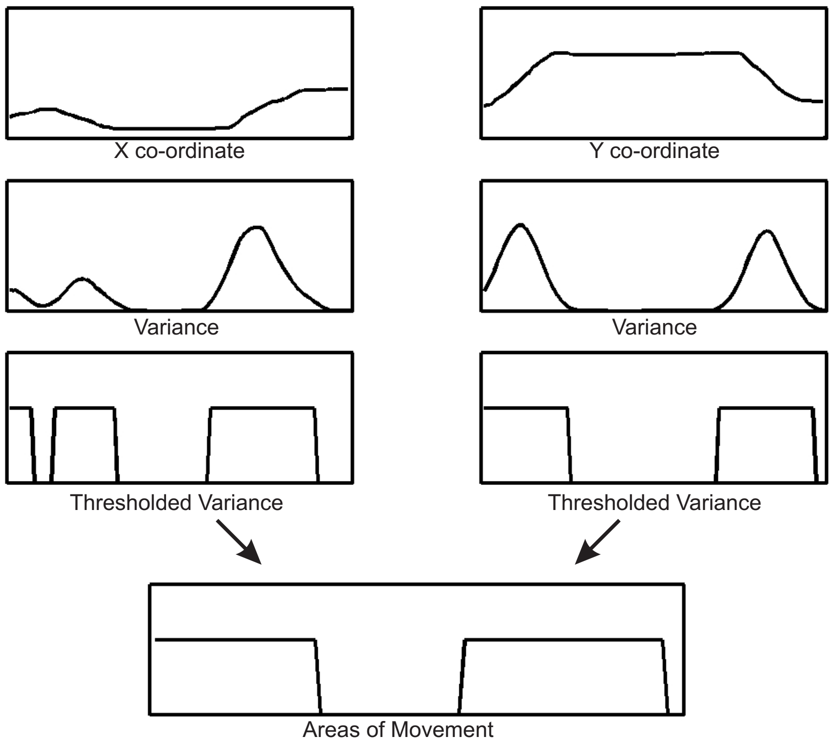

Figure 3.2: Determining areas of movement by calculating the thresholded variance for both co-ordinates and merging these results.

local neighbourhood. A result of this is that it will be marked as a movement. This is not desirable, as the movement should only start when the person is in the work environment, not before. This problem is illustrated in figure 3.4. The segmentation algorithm sets the *x* and *y* co-ordinates in the output data stream to $(-1, -1)$ when the person being tracked is not in the work environment. This means that it is relatively easy to fix this problem. All that needs to be done is to check for leading and trailing values of -1 on the movements. If they are present, they are removed.

A similar problem occurs when the person being tracked moves out the detection area. In this case, what would have been an area of minimal variance (no movement) is replaced with one of high variance. The result is that what should be defined as two separate movements is combined into one movement. This time,

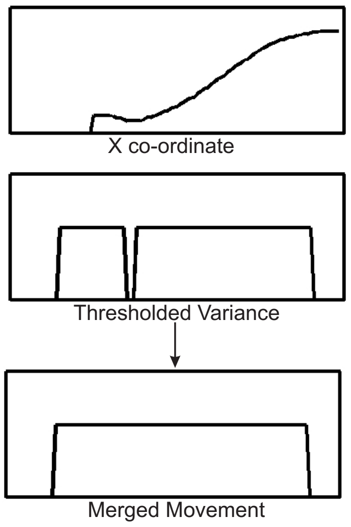

Figure 3.3: The two thresholded variance signals are merged because they are close to one another. This ensures that the movement does not get classified as two movements.

a search is done for values of $(-1,-1)$ inside the movement. If these values are found, then the movement is subdivided into the appropriate number of segments.

Sometimes there is a sharp edge in the data stream at a place that otherwise had minimal variance. This means that when the segmented area is divided into two movements, they must have their variance levels checked again. If this is not done, then a portion of the data stream in which there is no movement might be incorrectly classified. This is shown in figure 3.5 where no movements are present, but the edge effect of having left the work environment is present.

Two more checks are also done on all of the movements. These are:

- Making sure that their length is greater than a user-specified threshold (7 samples).
- Ensuring that there are no large discontinuities in the movement. This is also a heuristically-derived user-specified threshold.

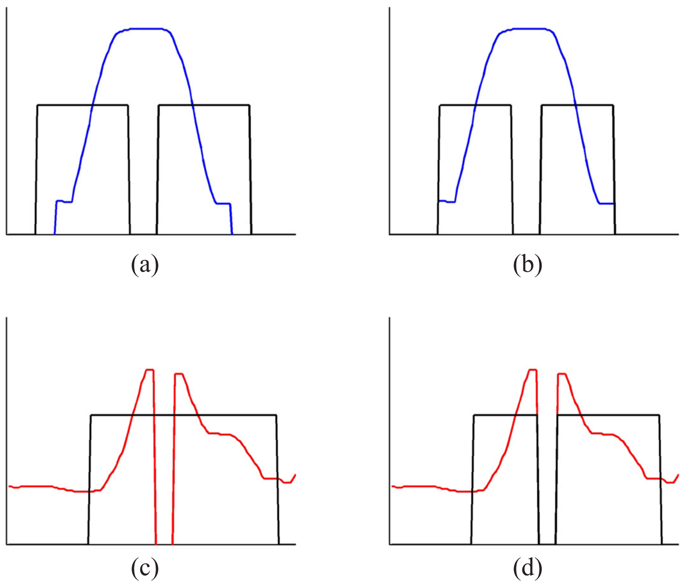

Figure 3.4: (a) Incorrect segmentation. The segmentation includes times when nobody is present; (b) correct segmentation of (a); (c) incorrect segmentation. There should be two separate movements; (d) correct segmentation of (c).

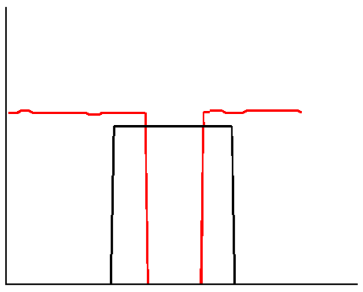

Figure 3.5: An example of how a false movement can be detected when the person being tracked is no longer in the 'detection area'.

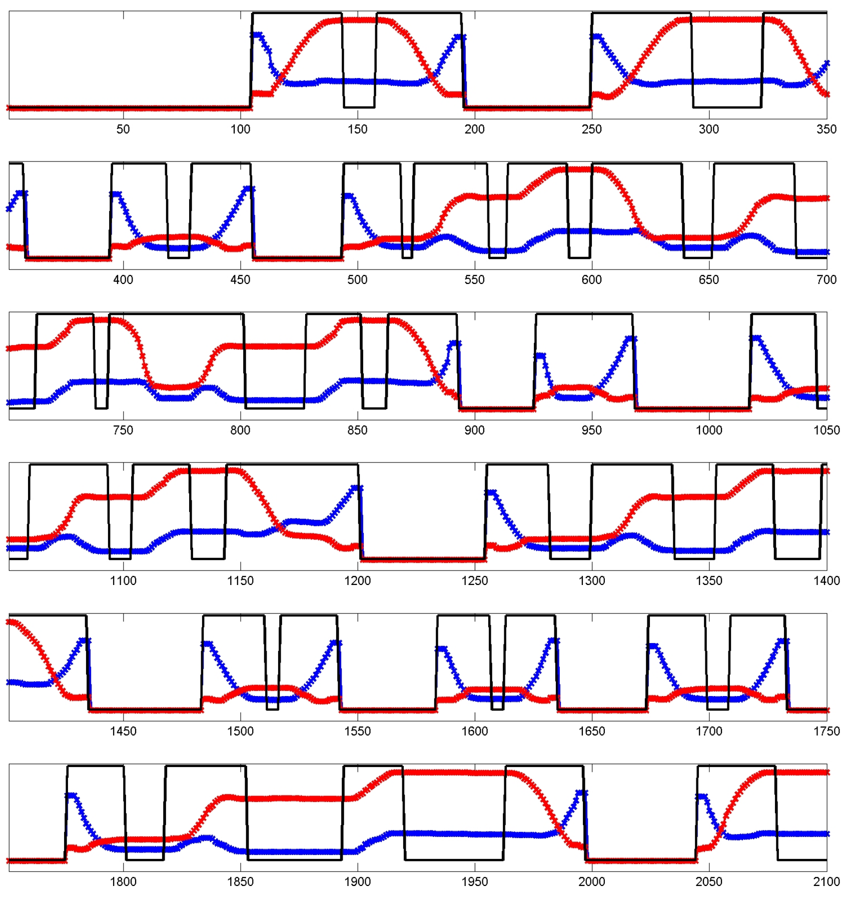

Figure 3.6: Segmentation of a large data set. The black indicates the individual movements after the segmentation has been performed. The red and blue indicate the person's *x* and *y* co-ordinates.

Chapter 4

Dynamic Time Warping

4.1 Distance measures

In order to be able to perform pattern recognition or classification, one needs to have a measure of similarity/dissimilarity between the two (or more) inputs that are being compared [3]. These inputs can take on various different forms: continuous signals, discrete signals, sequences or strings. There are many different distance measures used in various applications.

The euclidean distance is perhaps the most frequently used distance measure. It is given by the mathematical formula:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{k=1}^{i}(x_k - y_k)^2} \tag{4.1}$$

where **x** and **y** are vectors of dimensionality i. It is important that the relative sizes of the dimensions of the vectors are correctly chosen, otherwise the resulting distance measure could be meaningless. For instance, if the first dimension is measured in centimetres and the second dimension in millimetres, then clearly the second dimension will be more dominant in the resulting distance measure. This problem can be overcome by normalising the data before applying the distance measure, by using the weighted euclidean distance [3], or by using a different distance measure such as the mahalanobis distance.

Difficulties arise when these standard distance measures are applied to compare signals of different lengths. This situation occurs in various applications such as speech recognition, signature verification and the matching of people's movements. Since the signals are of different length, they are not members of the same vector space [29].

4.2 Linear time normalisation

If the two signals being compared are known to have a rate variation that is proportional to the duration of the signal, then linear time normalisation [1] (LTN) is a good method for determining an error measure between the two signals. One such method of performing linear time normalisation is to use the distance measure:

$$d(\mathbf{x},\mathbf{y}) = \sum_{i_x=1}^{T_x} d(i_x, i_y) \tag{4.2}$$

where i_x and i_y satisfy

$$i_y = \frac{T_y}{T_x} i_x \tag{4.3}$$

Because i_x and i_y are of integer value, it is implied that some kind of rounding off rule is applied. It is also possible to have the sum in equation 4.3 run from $i_y = 1$ to $i_y = T_y$ if desired [23].

The distance measure $d(i_x, i_y)$ in equation 4.2 can be any of the standard distance measures such as euclidean distance or mahalanobis distance.

If the rate of variation is not proportional to the length of the signal, then LTN does not perform well. This is shown in figure 4.2.

[1] Also known as linear time alignment

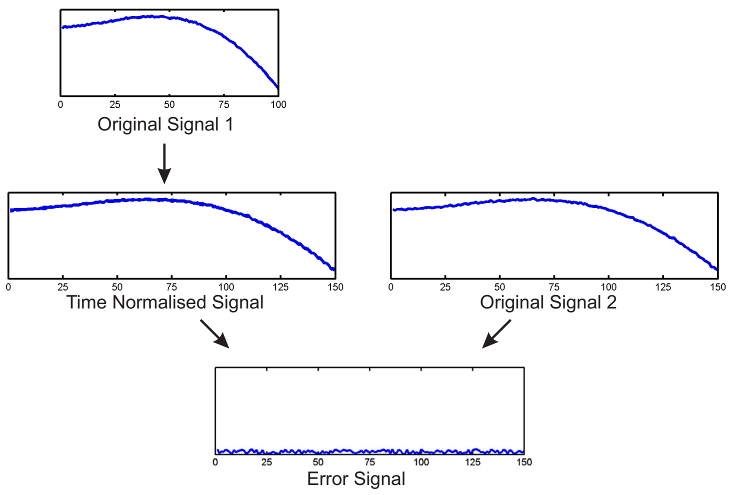

Figure 4.1: An example of linear time normalisation. The shorter input signal is converted so that it has the same length as the other input signal. The euclidean distance measure can then be used to find the error.

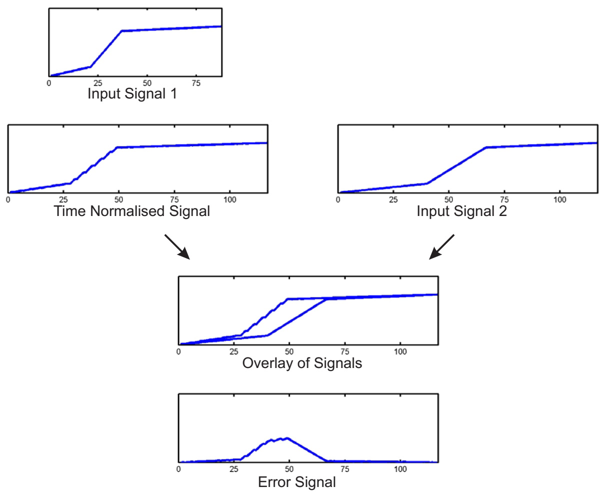

Figure 4.2: If the signals being compared do not have a rate of variation proportional to the length of the signal, then the resulting error signal will be large.

4.3 Dynamic time warping

Dynamic Time Warping (DTW) performs a more generalised time alignment than LTN. It makes use of two warping functions, ϕ_x and ϕ_y to warp the input data points onto a common time axis, k. If one defines the two input signals being compared as:

$$\mathbf{X} = (x_1, x_2, ...x_{T_x}) \tag{4.4a}$$

$$\mathbf{Y} = (y_1, y_2, ...y_{T_y}) \tag{4.4b}$$

then the warping functions, which relate the indices of the input signals (i_x, i_y) to the common time axis, k, are defined as:

$$i_x = \phi_x(k) \qquad k = 1, 2, ..., T \tag{4.5a}$$

$$i_y = \phi_y(k) \qquad k = 1, 2, ..., T \tag{4.5b}$$

Based on these definitions, a global dissimilarity measure, $d_\phi(\mathbf{X}, \mathbf{Y})$ can be defined as the accumulated distortion over the two signals:

$$d_\phi(\mathbf{X}, \mathbf{Y}) = \sum_{k=1}^{T} d(X_{\phi_x(k)}, Y_{\phi_y(k)}) m(k)/M_\phi \tag{4.6}$$

where $m(k)$ is a non-negative weighting co-efficient, M_ϕ is a path normalisation factor and $d(X_{\phi_x(k)}, Y_{\phi_y(k)})$ is a user-defined distance measure. Once again, the euclidean distance measure is usually used.

At this point the warping path, $\phi = (\phi_x, \phi_y)$, has not been defined. Clearly there are an extremely large number of possible paths that could be chosen. The desired warping path is the path that optimally aligns the two input signals. This will be the warping path that minimises the total accumulated distortion as defined in equation 4.6 [25]. The formula for the error value between two signals is given by the dynamic time warping formula:

$$d(\mathbf{X}, \mathbf{Y}) \equiv \min_{\phi} d_\phi(\mathbf{X}, \mathbf{Y}) \tag{4.7}$$

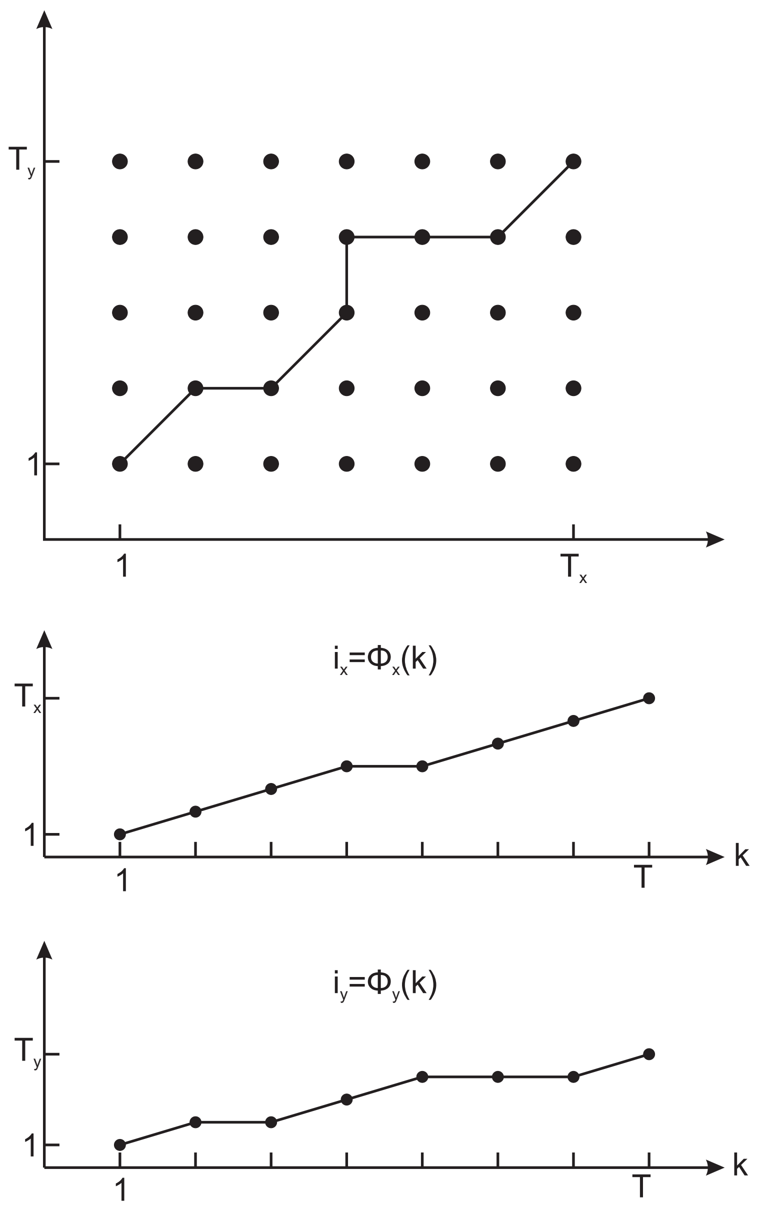

Figure 4.3: A warping path and its associated warping functions.

4.4 Constraints

The addition of a few logical constraints to the dynamic time warping problem helps to reduce the number of possible warping paths allowed [24]. Without the addition of these constraints, the minimisation would be able to conceivably result in a near-perfect match for two distinctly dissimilar input signals [23], thus rendering the comparison between the two signals useless.

4.4.1 Monotonicity constraints

Monotonicity constraints are perhaps the most fundamental of the constraints applied to the DTW. The monotonicity constraint maintains temporal order whilst aligning the two inputs. It eliminates the possibility of reverse warping along the time axis. It also implies that the slope of the warping path can never be negative [23]. It is specified as:

$$\phi_x(k+1) \geq \phi_x(k) \tag{4.8a}$$

$$\phi_y(k+1) \geq \phi_y(k) \tag{4.8b}$$

4.4.2 Endpoint constraints

In many cases, such as in speech recognition, the two inputs being compared have well defined endpoints that mark both the start and the end of the signal. In such a situation it makes sense to ensure that these endpoints are aligned and that warping takes place in between. The endpoint constraints for the warping function are of the form:

$$\phi_x(1) = 1 \qquad \phi_y(1) = 1 \tag{4.9a}$$

$$\phi_x(T) = T_x \qquad \phi_y(T) = T_y \tag{4.9b}$$

In some situations, there might be a certain degree of uncertainty about the accuracy of the start and endpoints of the signals. In such circumstances, the endpoint constraints can be relaxed to take this into account.

4.4.3 Local continuity constraints

Local continuity constraints ensure proper time alignment while keeping the loss of information to a minimum [23]. Local continuity constraints also help to ensure that excessive compression or expansion of the time scales is avoided [22]. They are a more highly constrained version of the monotonicity constraints, as they specify the set of movements that are allowed to be made in the warping lattice. Local continuity constraints are based on heuristics. This means that there is no way of determining the optimal local continuity constraints other than performing tests on various options so as to see which one performs the best. One common local continuity constraint proposed (see figure 4.4a) by Sakoe and Chiba [25] is defined as:

$$0 \geq \phi_x(k+1) - \phi_x(k) \leq 1 \tag{4.10a}$$

$$0 \geq \phi_y(k+1) - \phi_y(k) \leq 1 \tag{4.10b}$$

Such definitions of local continuity constraints are often difficult to interpret, so they are often shown graphically or defined as a set of incremental path movements, P.

The distinction should be made between symmetrical and asymmetrical local continuity constraints. The former allows for the warping of both input signals to a common time axis, whereas the latter only allows for the warping of one input signal onto the other. Sakoe and Chiba [25] have shown that the symmetrical form is superior to the asymmetrical form.

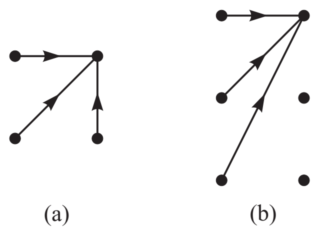

Figure 4.4: (a) Symmetrical local continuity constraints; (b) asymmetrical local continuity constraints [30].

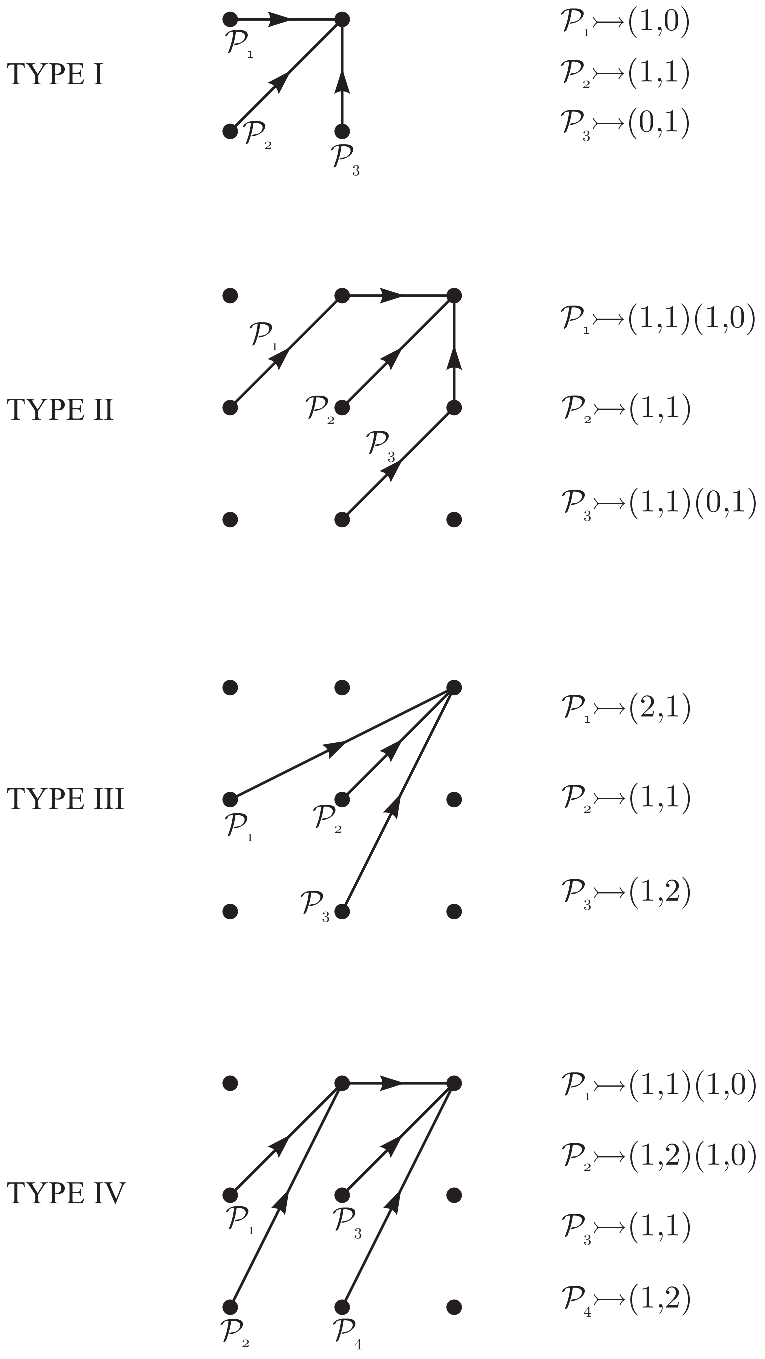
TYPE I
$\mathcal{P}_1 \rightarrowtail (1,0)$
$\mathcal{P}_2 \rightarrowtail (1,1)$
$\mathcal{P}_3 \rightarrowtail (0,1)$
TYPE II
$\mathcal{P}_1 \rightarrowtail (1,1)(1,0)$
$\mathcal{P}_2 \rightarrowtail (1,1)$
$\mathcal{P}_3 \rightarrowtail (1,1)(0,1)$
TYPE III
$\mathcal{P}_1 \rightarrowtail (2,1)$
$\mathcal{P}_2 \rightarrowtail (1,1)$
$\mathcal{P}_3 \rightarrowtail (1,2)$
TYPE IV
$\mathcal{P}_1 \rightarrowtail (1,1)(1,0)$
$\mathcal{P}_2 \rightarrowtail (1,2)(1,0)$
$\mathcal{P}_3 \rightarrowtail (1,1)$
$\mathcal{P}_4 \rightarrowtail (1,2)$

Figure 4.5: An example of various local continuity constraints [23].

4.4.4 Global path constraints

Because of the introduction of local continuity constraints, there are large areas of the (i_x, i_y) plane that are excluded from the set of possible paths that the warping path may take. The area of allowable paths is usually in the shape of a parallelogram with vertices at $(1,1)$ and (T_x, T_y). The other vertices are defined by the local continuity constraints. The slopes of the parallelogram depend on the type of local continuity constraints being used. They are typically found to be either $(2\ ,\ {}^1\!/\!_2)$ or $(3\ ,\ {}^1\!/\!_3)$.

Sakoe and Chiba have have proposed a further global path constraint [25]:

$$|\phi_x(k) - \phi_y(k)| \leq T_0 \tag{4.11}$$

T_0 represents the maximum allowable time deviation between the two patterns at any moment [23]. This constraint also helps to prevent any excessive stretching or compressing of the signals during the comparison. It also helps to limit the search space for a suitable warping path, ϕ.

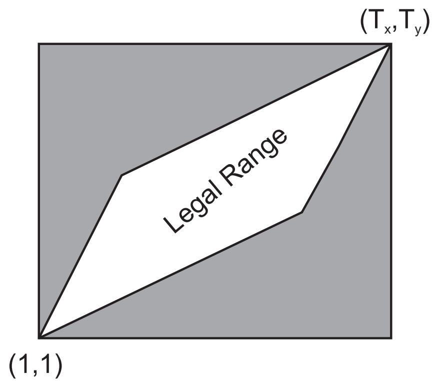

Figure 4.6: An example of a global path constraint without Sakoe and Chiba's additional constraint (Type II).

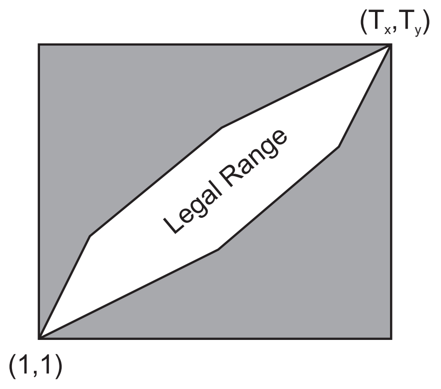

Figure 4.7: An example of a global path constraint with Sakoe and Chiba's additional constraint (Type II).

4.4.5 Slope weighting

The slope weighting constraint is denoted by $m(k)$ (see equation 4.6). The slope weighting can be used to set a preference for the different possible movements as specified in the local continuity constraints. As with local continuity constraints, there are many heuristic slope weighting functions that have been proposed. A larger value of slope weight indicates less preference for that path to be taken.

The path normalisation factor, $M_\phi(k)$ (see equation 4.6), is incorporated into the DTW formula in order to provide normalisation such that the total accumulated distortion is independent of the length of the two signals being compared. Because of this, it is usually set to be the sum of the slope weighting coefficients:

$$M_\phi = \sum_{k=1}^{T} m(k) \tag{4.12}$$

Depending on the slope weighting coefficients used, the path normalisation factor is often equal to either T_x or $T_x + T_y$. This means that the path normalisation factor is a constant and can be taken out of the summation in equation 4.6 resulting in:

$$d_\phi(\mathbf{X},\mathbf{Y}) = \frac{1}{M_\phi}\sum_{k=1}^{T} d(X_{\phi_x(k)}, Y_{\phi_y(k)})m(k) \tag{4.13}$$

and an updated version of the minimisation equation 4.7:

$$d(\mathbf{X},\mathbf{Y}) \equiv \frac{1}{M_\phi} \min_\phi \sum_{k=1}^{T} d(X_{\phi_x(k)}, Y_{\phi_y(k)}) m(k) \tag{4.14}$$

The path normalisation factor is often set to either T_x or $T_x + T_y$ even when this is not the correct value. This is because having the path normalisation factor as a constant value makes the minimisation problem of equation 4.7 much easier to solve [23]. This can lead to a bias in the type of warping path followed.

4.5 Dynamic programming

In order to find the optimal warping path, it is necessary to solve the minimisation problem in equation 4.14. If one were to use a standard recursive implementation to solve the minimisation, it would be of the order N^2M^2 [29], where N and M are the lengths of the two input signals.

A dynamic programming solution to the minimisation problem can be implemented in order NM [29]. The execution time of the minimisation problem is dependent on the local and global path constraints used. The smaller the parallelogram in the search lattice, the quicker the execution time. The principle of dynamic programming was created by Richard Bellman [5], which was based on his principle of optimality:

> *An optimal policy has the property that whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with respect to the state which results from the initial decision.*

Cooper and Cooper [5] have simplified the principle of optimality to the following:

> *Every optimal policy consists only of optimal sub-policies.*

The minimal partial accumulated distortion can be defined for the optimal path from $(1,1)$ to (i_x, i_y) as:

$$D(i_x, i_y) \equiv \min_{\phi, T'} \sum_{k=1}^{T'} d(X_{\phi_x(k)}, Y_{\phi_y(k)}) m(k) \tag{4.15}$$

where

$$\phi_x(T') = i_x \tag{4.16a}$$

$$\phi_y(T') = i_y \tag{4.16b}$$

Implementing the principle of optimality, this means that:

$$D(i_x, i_y) = \min_{i'_x, i'_y} [D(i'_x, i'_y) + \zeta((i'_x, i'_y), (i_x, i_y))] \tag{4.17}$$

where ζ is the weighted local distance between point (i'_x, i'_y) and (i_x, i_y). It can also be expressed in terms of the local continuity constraints. For example, if TYPE I constraints were being used, then the minimal partial accumulated distance from $(1,1)$ to (i_x, i_y) can be expressed as:

$$D(i_x, i_y) = min \begin{bmatrix} D(i_x, i_y - 1) + d(i_x, i_y) \\ D(i_x - 1, i_y - 1) + d(i_x, i_y) \\ D(i_x - 1, i_y) + d(i_x, i_y) \end{bmatrix} \tag{4.18}$$

The algorithm operates by starting at the point $(1,1)$ in the lattice of possible warping paths. The lattice is traversed by moving up each column incrementally, and then moving on to the next column. In this way each point in the lattice is visited just once. Also, when it is visited, the minimal partial accumulated distance for that point can be calculated, because the partial distance for all of its legal predecessors will already have been calculated. When the endpoint (T_x, T_y) is reached, the total accumulated distortion will have been calculated. This can then be divided by the path normalisation constant, M_ϕ, to arrive at the final error between the two signals.

If it is desired to know what the optimal warping path is, it is necessary to keep a back-pointer for each point in the lattice. This pointer indicates which predecessor was optimal. These pointers can be traced backward from (T_x, T_y) to $(1,1)$ to find the optimal warping path.

4.6 Considerations in dynamic time warping

Various considerations have been suggested by Rabiner and Juang [23] and Rabiner *et al.* [24] with regard to optimising the DTW algorithm for specific conditions.

4.6.1 Relaxed endpoint constraints

If the endpoint constraint is relaxed, then it is possible to find better matches between the inputs when the segmentation of the signals is poor. The resulting 'optimal match' between the two signals may not contain all of the sample values for each of the inputs. The disadvantage of relaxing the endpoint constraints is that the computational expense is increased. This is because the number of points in the lattice that need to be checked is larger.

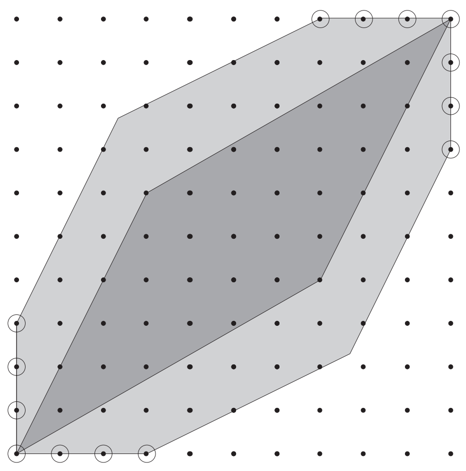

Figure 4.8: Relaxed endpoint constraints. The dark grey shows possible warping paths with endpoint constraints. The light grey shows possible warping paths with relaxed endpoint constraints. The ringed points show possible start and endpoints.

4.6.2 Localised range constraints

Rabiner *et al.* [24] also refer to this technique as UELM (unconstrained endpoints, local minimum). This method is used to reduce the computation time by only allowing warping paths that are close to the instantaneous locally minimum warping path. This is shown in figure 4.9. This algorithm does not guarantee that the optimal path will be found.

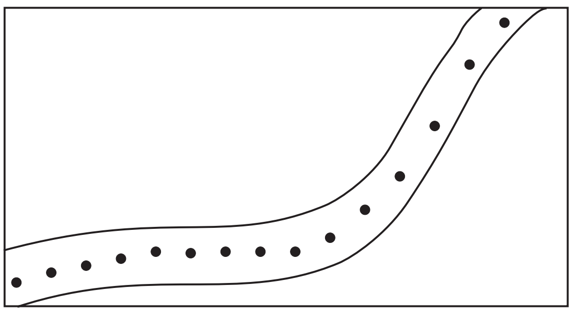

Figure 4.9: Allowable warping path region with localised range constraints [23].

4.6.3 Multiple time alignments

It has been suggested [23] that it might be more meaningful to look at a certain number of best paths. This is useful when the single best path might be too sensitive or not robust enough. By analysing the K-best paths, better results may be obtained. Serial and parallel algorithms are available to find the K-best paths [23].

Chapter 5

Optimising the DTW

In order to successfully make use of the dynamic time warp to compare two different movements, the generic form of the DTW needs to be optimised so that its output is useful and meaningful. This chapter presents the various ways in which the DTW was optimised for determining the similarity between individual movements.

5.1 Distance measure

It was decided that the euclidean distance measure would be used in the DTW to compare the various sample points. This is because the euclidean distance measure is widely used in many applications and is known to perform well.

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{k=1}^{i}(x_k - y_k)^2} \tag{5.1}$$

Since this was a prototype system, the computational requirements of the euclidean distance measure were ignored. However, if more speed is required, other distance measures (which are optimised for speed) can be used. The squared euclidean distance is a good example of this. It can be calculated quicker as no square root operation needs to be performed. The other effect of the squared euclidean distance is that it places more emphasis on points that are further apart from each other. The formula for squared euclidean distance is:

$$d(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{i} (x_i - y_i)^2 \tag{5.2}$$

Initial work was done by performing two individual DTWs, one for the set of x co-ordinates and one for the set of y co-ordinates for the movements being compared. It was expected that the warping paths for each of these calculations would be similar. The errors for the two paths were then added together to determine the total error between the two movements.

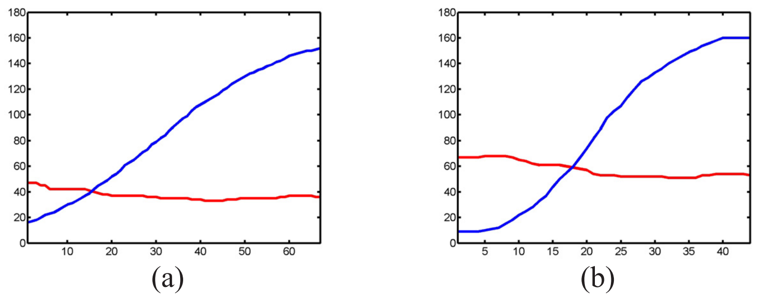

Figure 5.1: Two movements being compared in figure 5.2.

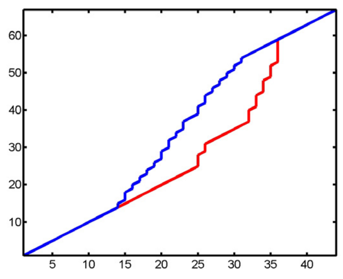

Figure 5.2: The optimal warping paths for the x and y co-ordinates of two movement patterns.

As can be seen from figure 5.2 the warping paths for the x and y co-ordinates differ quite substantially. For this reason, it was decided that it would be better to match the *x* and *y* co-ordinates simultaneously by using a two-dimensional distance measure in the DTW. This means that the optimal warping path is optimal in terms of both sets of co-ordinates and is therefore a more reliable measure of the difference between the two signals.

5.2 Local continuity constraints

The local continuity constraints that have been suggested by Sakoe and Chiba [25] are optimised for use in speech recognition applications. This does not however mean that they are optimal for determining a relevant distance measure between two movements. In particular one of the assumptions of their local continuity constraints is that the lengths of the two signals being compared are very similar.

Although in theory, with a local continuity constraint that has a maximum slope of 2 (and minimum slope of 1/2), one can effectively find the best warping path for two signals one of which is twice the length of the other, this is not really the case. This is shown in figure 5.3.

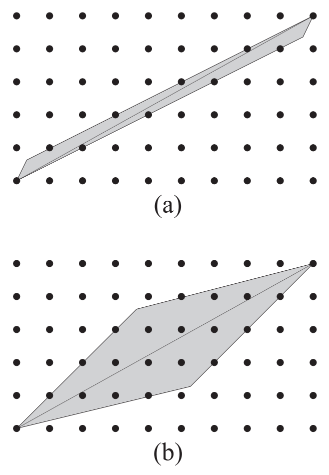

Figure 5.3: (a) Possible warping paths with a maximum slope of 2; (b) Possible warping paths with a maximum slope of 1. The dotted line indicates the linear warping path.

If local continuity constraints which have a maximum slope value of 2 (and minimum slope of 1/2) are used, then the number of possible warping paths that can be utilised is minimal. Many warping paths which would be suitable are not allowed. This means that the likelihood of obtaining a useful output from the DTW is not very high. The image in figure 5.3(b) shows the possible warping paths with a maximum slope of 1 and a minimum slope of 1/4. As can be seen this is a better setup. Many more suitable warping paths can be assessed and the likelihood of a more suitable output value is higher.

Since the duration of two similar movements can be quite different from one another, new local continuity constraints need to be introduced. A set of numerous movements was collected. The lengths of similar movements were compared to determine the maximum ratio between the lengths of similar movements. A histogram showing the ratios of the lengths of individual movements to the shortest length similar movement is shown in figure 5.5.

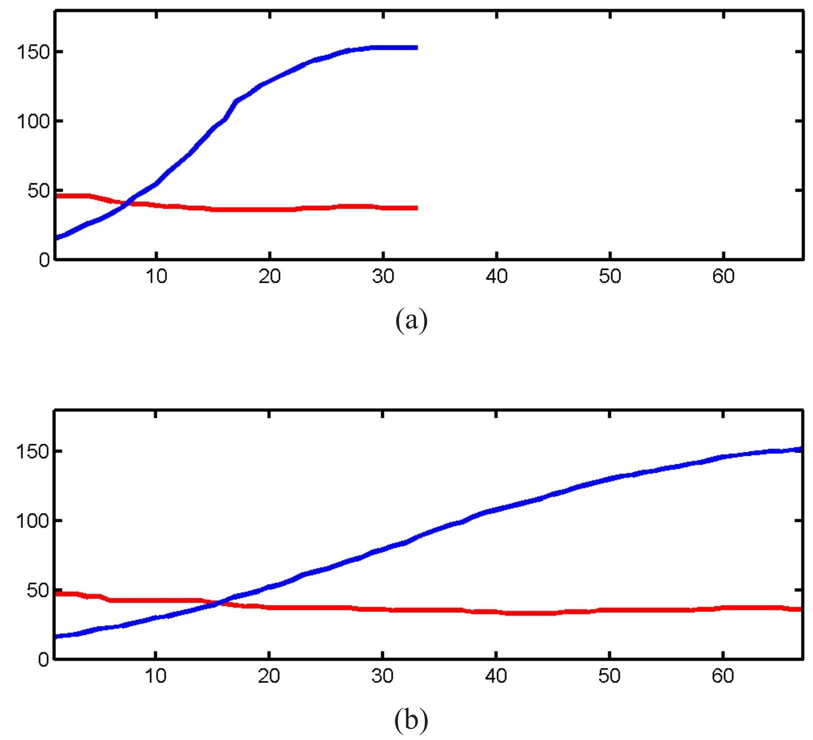

Figure 5.4: Two signals that represent the same movement. However, the durations of the movements differ by a large amount.

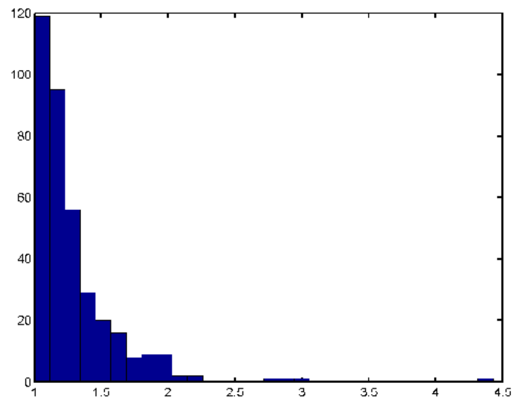

Figure 5.5: Histogram showing ratios of signal length to length of smallest similar signal.

As can be seen in the histogram (figure 5.5), most movements (82%) differ in length at most by a factor of 1.5, while very few (2%) differ in length by a factor of greater than 2. Based on these numbers, the following local continuity constraints were devised:

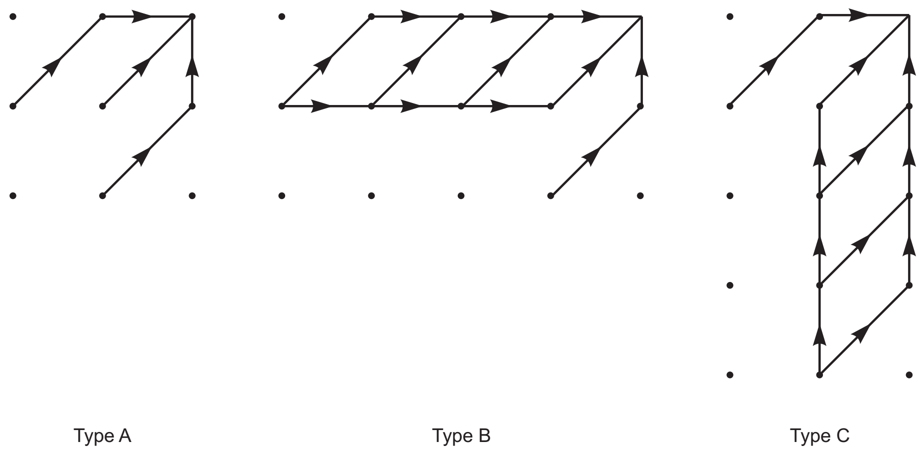

Figure 5.6: The local continuity constraints that were devised.

These various local continuity constraints would be implemented depending on the ratio of the lengths of the two input signals.

Input Signal Length Ratio	Local Continuity Constraint Type
$> \frac{4}{15}$ and $< \frac{3}{4}$	B
$\geq \frac{3}{4}$ and $\leq \frac{3}{2}$	A
$> \frac{3}{2}$ and $< \frac{15}{4}$	C

By using these three different sets of local continuity constraints, signals that have a length ratio that is between $\frac{4}{15}$ (≈ 0.27) and $\frac{15}{4}$ (3.75) can be compared using the DTW. Signals that have length ratios that fall outside this range are considered to be too extreme to have a meaningful distance measure between them. Their distance is set to an arbitrarily high value that indicates that the two signals are highly dissimilar.

5.3 Endpoint constraints

Since the segmentation of the movement data stream into individual movements is done automatically, it cannot be guaranteed that the start and endpoints of the individual movements are perfectly aligned. By relaxing the endpoint constraints, the DTW can effectively find a signal within another one. The degree to which this can be achieved can be varied by adjusting the degree to which the endpoint constraints are relaxed.

Two different methods of relaxing the endpoint constraints were investigated. In the first method (figure 5.7a), the endpoints of only one signal were relaxed. This meant that the whole of the one signal had to be found in the other one. The second method (figure 5.7b) involved relaxing the endpoint constraints on both of the input signals.

Care must be taken when relaxing the endpoint constraints. If they are relaxed too much, then highly dissimilar signals can still be given a very low dissimilarity score. An example of this is shown in figure 5.8.

The relaxed endpoint constraints as shown in figure 5.7(b) were used in the final implementation. This is because the order in which the signals being compared are presented to the DTW is not important. This is equivalent to:

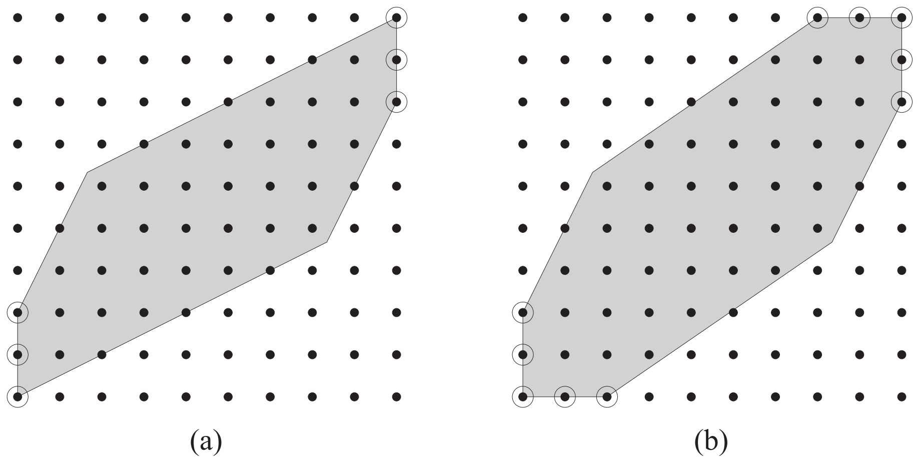

Figure 5.7: The two different endpoint constraints types and their associated possible warping paths. The ringed lattice points indicate possible start and endpoints.

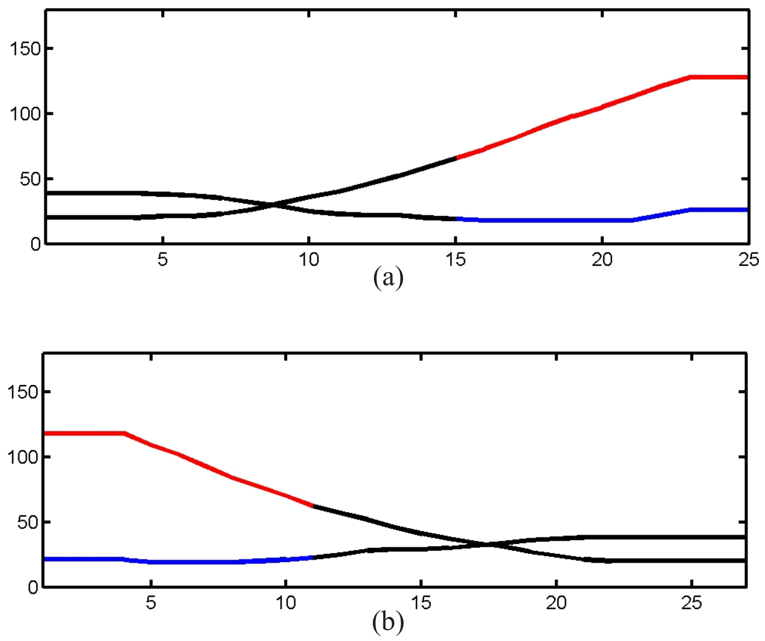

Figure 5.8: Two signals being compared with endpoint constraints as in figure 5.7b. The black areas indicate the regions that have been matched. Although the signals are highly dissimilar, a reasonable match is found.

$$DTW(a,b) = DTW(b,a) \tag{5.3}$$

This is not the case for the relaxed endpoint constraints shown in figure 5.7(a).

5.4 Slope weighting

Many different slope weighting functions have been suggested [23], however, there is no guarantee that any of them are optimal for any given problem. Because of this, it was decided to implement a simple slope weighting scheme. All of the slope weights were set to a value of one. This means that no preference is made to any of the possible moves that can be made through the lattice. This is reasonable since all of the possible moves (as specified in the local continuity constraints) do not skip out any intermediate points in the lattice. If local continuity constraints are such that they skip lattice points and they do not have higher slope weightings, then there will be a much greater tendency for these paths to be found in the optimal warping path.

This is shown in figure 5.9. Path *A* consists of two movements, A_1 and A_2, while path *B* consists of just one movement. Both paths have the same start and end-points. If path *B* did not have a slope weighting of 2, it is highly likely that the distance measure following path *B* would be approximately half the distance measure of following path *A*. For this reason path *B* is given a higher slope weight.

5.5 Global path constraints

No global path constraints were added to the DTW other than those caused by the local continuity constraints. This was done so as to be able to match signals that might have large variances in their rate of movement. For example when someone starts walking fast and then slows down drastically. This will result in a warping path that is quite far from the linear time warp path. The additional global path constraints suggested by Sakoe and Chiba [25] limit the amount of

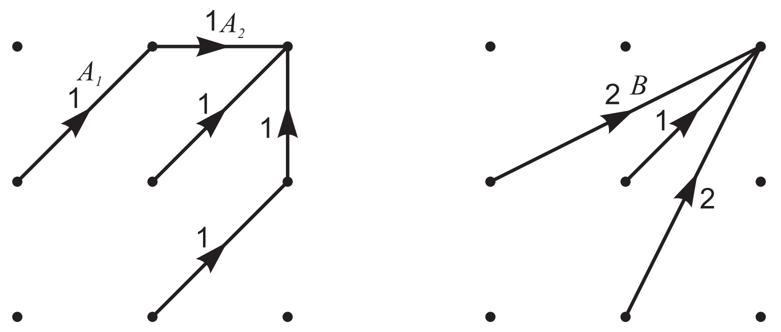

Figure 5.9: Two similar local continuity constraints and their associated path weightings.

deviation between the two signals at any time. An example of two signals where the deviation is quite high is shown in figure 5.10. It can be clearly seen that there is a large change in speed of movement in the upper signal, whereas the lower signal experiences a more constant speed of movement. This results in the warping path deviating greatly from the linear warping path.

5.6 False negatives and false positives

If the algorithms developed in this work are to be implemented in an online security system, it will be necessary to have a target for the allowable number of false negatives and false positives allowed. It is highly likely that both of these values may not be achieved and that it will be necessary to decide on a suitable tradeoff.

A false positive is any movement that was identified as being similar to another movement when in fact it was not.

A false negative is any movement that is similar to another movement, but was not identified as similar by the system.

A high false positive rate will alert the security personnel to many events, most of which will be normal movements. This is generally more effective than having a high false negative rate in which many abnormal movements are not brought to the security personnel's attention as they are thought to be normal.

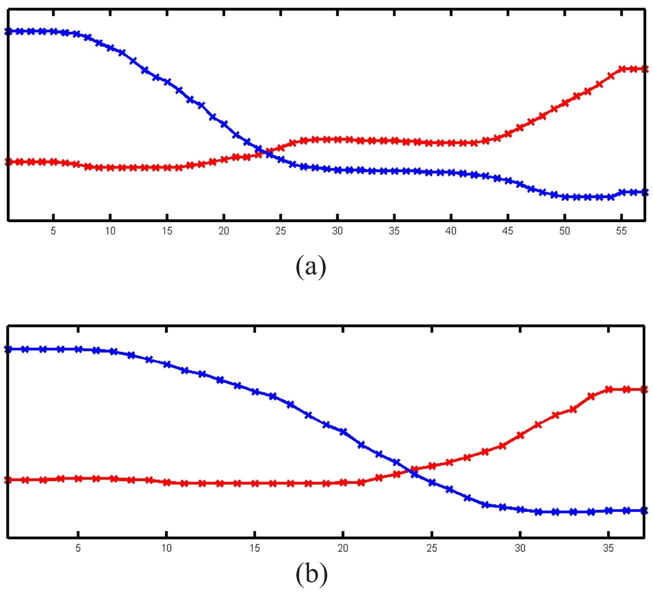

Figure 5.10: Signals with differing degrees of speed of movement.

By knowing the desired level of false positives and false negatives before the implementation of a final system, the global and local path constraints can be suitably adjusted. For example, setting the allowable warping region to a very small area will result in only highly similar movements being considered similar.

Chapter 6

Clustering

The application of clustering is found in many different scientific fields, including life sciences, statistics, mathematics and computer science [26]. The objective of clustering is to partition a set of items into various subsets. Each of these subsets is called a cluster. It is desired that the partitions formed are such that the resulting clusters are homogeneous and well separated. Mathematically, the data can be represented as a set of n objects:

$$O = \{o_1, o_2, ..., o_n\} \tag{6.1}$$

The clusters can then be defined as:

$$C = \{c_1, c_2, ..., c_k\} \tag{6.2}$$

where k is the number of clusters present and the following constraints apply:

$$c_1 \cup c_2 \cup c_3 ... \cup c_k = C \tag{6.3a}$$

$$c_i \cap c_j = \phi \quad \forall\, i \neq j \tag{6.3b}$$

This means that each object must be a member of only one cluster. It is desired that objects that are part of the same cluster have a high degree of natural association, and that those which are from different clusters have a low degree of natural association. How this natural association is measured is dependent on the similarity measure (distance measure) used. As with most problems that use a distance measure, the choice of distance measure can affect the outcome of the process. The euclidean distance measure is a popular choice [11]:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{k=1}^{i}(x_k - y_k)^2} \tag{6.4}$$

As mentioned in section 4.1, it is important to ensure that the dimensions of the data are in proportion to one another so that one dimension does not dominate the others. It is also important to ensure that one uses all of the known information about the problem. Extracting useful features is also important. Jain *et al.* point out that choosing the wrong co-ordinate system can make results incorrect. The semi-circle in figure 6.1 is likely to be fragmented if one were to use Cartesian co-ordinates. Polar co-ordinates on the other hand, would all have a very similar radius value, resulting in a better clustering solution [11].

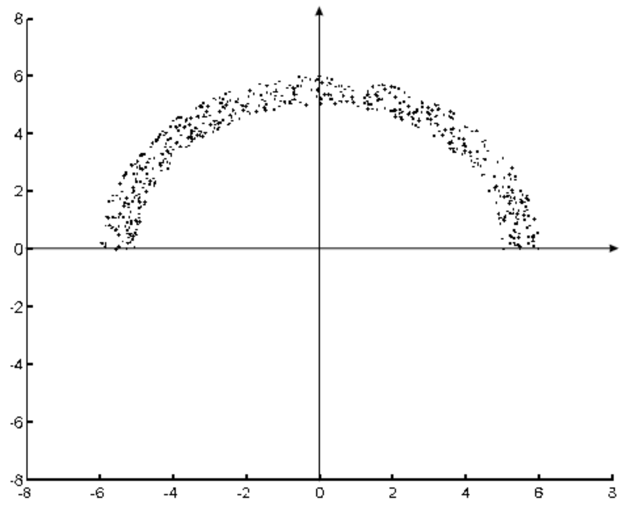

Figure 6.1: Data which would best be represented using polar co-ordinates.

Jain *et al.* [11] have described the various different clustering techniques available, and have come up with the taxonomy of clustering approaches that is shown in figure 6.2.

Clustering problems can be further divided in two: those in which the number of

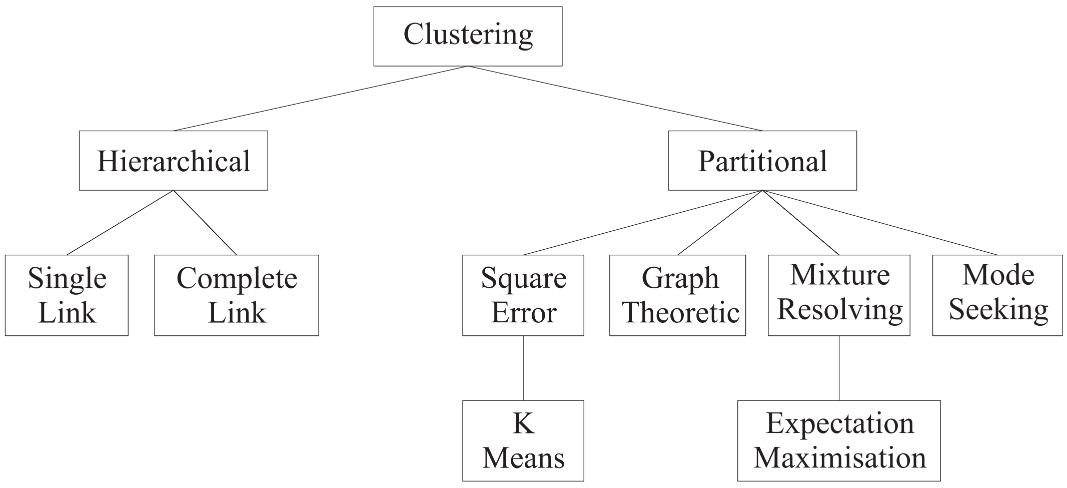

Figure 6.2: A taxonomy of clustering approaches [11].

clusters is known before the clustering process takes place, and those in which the number of clusters is unknown. The methods that will be discussed are those which can be used for the latter case. This is because the number of clusters is unknown in this work.

6.1 Hierarchical clustering

The solution to the hierarchical clustering problem generally takes the form of a dendrogram. This is a tree type structure that indicates how the individual objects are clustered together. Hierarchical clustering is usually done in either an agglomerative (bottom-up) or a divisive (top-down) manner.

6.1.1 Agglomerative clustering

All of the objects are divided into n clusters, one for each object. The next step is to reduce the number of partitions to $n-1$. To achieve this, the two nearest clusters are merged together. This process is repeated until there is only one cluster. The definitions for exactly how the clusters are merged and which distance measure is used to calculate the closest clusters can take on a large variety of forms. Depending on the methods used, the outcome of the clustering may vary.

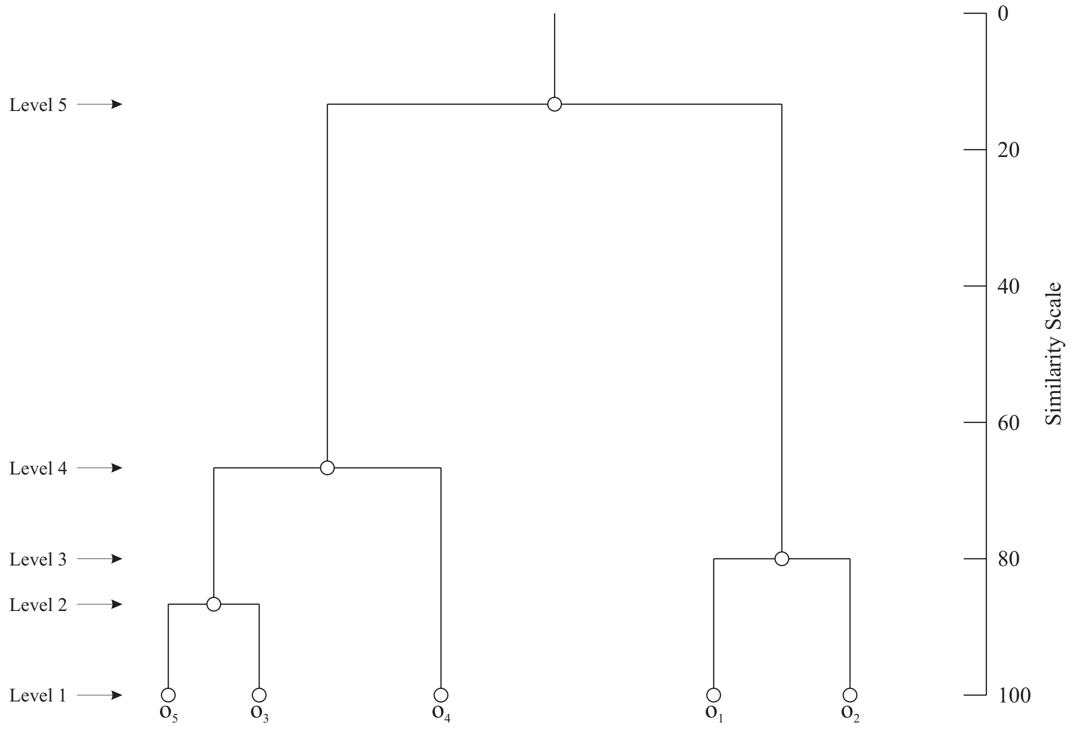

Figure 6.3: A typical dendrogram for hierarchical clustering.

6.1.2 Divisive clustering

Divisive clustering is similar to agglomerative clustering. The difference being that instead of starting with all of the objects as clusters, the algorithm begins with one cluster containing all of the objects. This cluster is then split repeatedly until only clusters that contain one object remain. The computation of divisive clustering is usually more complicated than that of agglomerative clustering [6].

6.1.3 Finding divisions

Once the dendrogram has been created, the final clusters can be determined in two ways. If the desired number of clusters is known, then the objects that fall into these clusters can be easily determined. For example, dividing the data in figure 6.3 into two clusters results in:

$$c_1 = \{o_3, o_4, o_5\}$$

$$c_2 = \{o_1, o_2\}$$

while dividing the data into three clusters results in:

$$c_1 = \{o_3, o_5\}$$

$$c_2 = \{o_1, o_2\}$$

$$c_3 = \{o_4\}$$

If, however, the desired number of clusters is unknown, the task is more difficult. One way to determine the number of clusters is to look at the inconsistency value for each of the links. The inconsistency value is an indication of how similar in length a link is to those links beneath it. If the lengths of the links are similar, it indicates that the objects are likely to be of the same type. If they differ greatly, then it is likely that the link forms a natural division in the data set [10].

The cophenetic correlation coefficient can be used to determine whether the dendrogram fits the data well or not [8]. This is computed by finding the element-wise correlation of Z with d. Z is defined as the cophenetic matrix, where each element Z_{ij} contains the level at which o_i and o_j are in the same branch of the dendrogram. The distance matrix, d, contains the distance between o_i and o_j at location d_{ij}.

The higher the value of the cophenetic correlation coefficient, the better the fit of the dendrogram. It has been suggested that the value of the cophenetic correlation coefficient should be greater than 0.8 for one to assume that the data are not just one big cluster [8].

6.2 Graph theoretic methods

In many clustering algorithms the order in which the objects are presented to the clustering algorithm can affect the outcome of the method. Graph theoretic methods were introduced to try to eliminate this problem by processing all of the samples simultaneously [3]. The downside of graph theoretic methods is that they can result in an increase in computational time [3].

6.2.1 Similarity matrix

A similarity matrix is used to show the similarity between all of the objects to be clustered. Firstly, a definition of similarity is required. Two objects are considered similar if:

$$d(o_i, o_j) \leq \theta \tag{6.5}$$

As usual, it is up to the user to provide a suitable distance function, $d(o_i, o_j)$. θ is a user-specified threshold value. The similarity matrix, S, is then defined as the $N \times N$ matrix (where N is the number of objects) whose individual elements are given by:

$$s_{ij} = \begin{cases} 1, & d(o_i, o_j) \leq \theta \\ 0, & d(o_i, o_j) > \theta \end{cases} \qquad i, j = 1, 2, \ldots, N \tag{6.6}$$

Once the similarity matrix has been created, the following algorithm can be used to determine the number of clusters:

Repeat:

1. Determine the row in S with the maximum number of 1's in it. If there is more than one row with the maximum number of 1's, choose any row. Call this row i.

2. Form a cluster with object o_i and all objects that are similar to o_i. That is, add object o_j to the cluster if $s_{ij} = 1$.

3. Add all objects that are similar to the objects already in the cluster to the cluster. That is, for all objects o_j in the cluster, add o_k if $s_{jk} = 1$.

4. Remove all the columns and rows that correspond to the objects in the cluster from the similarity matrix.

until no more clusters can be formed.

The choice of θ is a vital one, as it will effectively determine the number of clusters that will be formed. The larger it is, the fewer the clusters generated. It is also important to notice that the similarity matrix holds a total of N^2 elements. This can cause computational difficulties when N is large.

As with most clustering techniques, a few simple changes can be made to change how the clustering works. Two examples for how this can be done when using a similarity matrix are:

Complete-link similarity matrix

If one removes step 3 from the previous algorithm, then it ensures that all of the points in each cluster are within the specified threshold distance, θ, from each other.

Single-link similarity matrix

If one modifies step 3 of the previous algorithm to:

> Repeatedly add all objects that are similar to the objects in the cluster to the cluster. Do this until no more objects are added.

then the clustering performed will be a single linkage. This means that the clusters will be long and 'stringy' as opposed to 'clumpy'.

6.2.2 Minimal spanning trees

A minimal spanning tree is defined [3]:

> *The minimal spanning tree is that spanning tree of minimal weight (amongst all possible spanning trees).*

The following definitions are useful [3]:

1. An *edge* is a connection between two points.
2. The *weight* of an edge is the distance between the two points it connects.
3. A *path* is a sequence of edges connecting two points.
4. A *loop* is a closed path.
5. A *connected graph* has one or more paths between any pair of points.
6. A *tree* is a connected graph with no closed loops.
7. A *spanning tree* is a tree that contains every point.
8. The *weight* of a tree is the sum of the weights assigned to each edge.

Clusters are created by removing edges that have a value greater than a user-specified threshold. For example, see figure 6.4 where the threshold has been set at a value of 5, resulting in three clusters being formed.

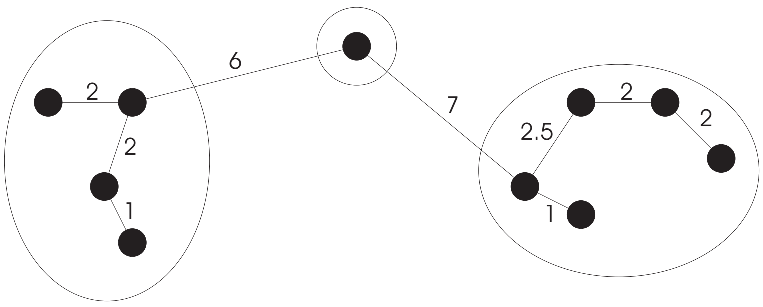

Figure 6.4: A minimal spanning tree and its clusters.

6.2.3 Batchelor and Wilkins' algorithm

This algorithm is based on the furthest neighbour clustering technique. It works as follows [3]:

1. Select a random object, and make it the centre of cluster number 1.
2. Select the object furthest from this cluster, and make that the centre of cluster 2.
3. Calculate the distances to all of the objects to the clusters.
4. Determine the minimum distance to a cluster for each object.
5. Chose the maximum of these minimum distances.
6. If this distance is greater than a fraction of the typical distance between the clusters, then let this be the centre of a new cluster.
7. Repeat steps 3 to 6, until no new clusters are formed.
8. Classify objects according to their nearest cluster.

This algorithm was modified by changing step 6 to:

6. If this distance is greater than a user-specified threshold, then let this be the centre of a new cluster.

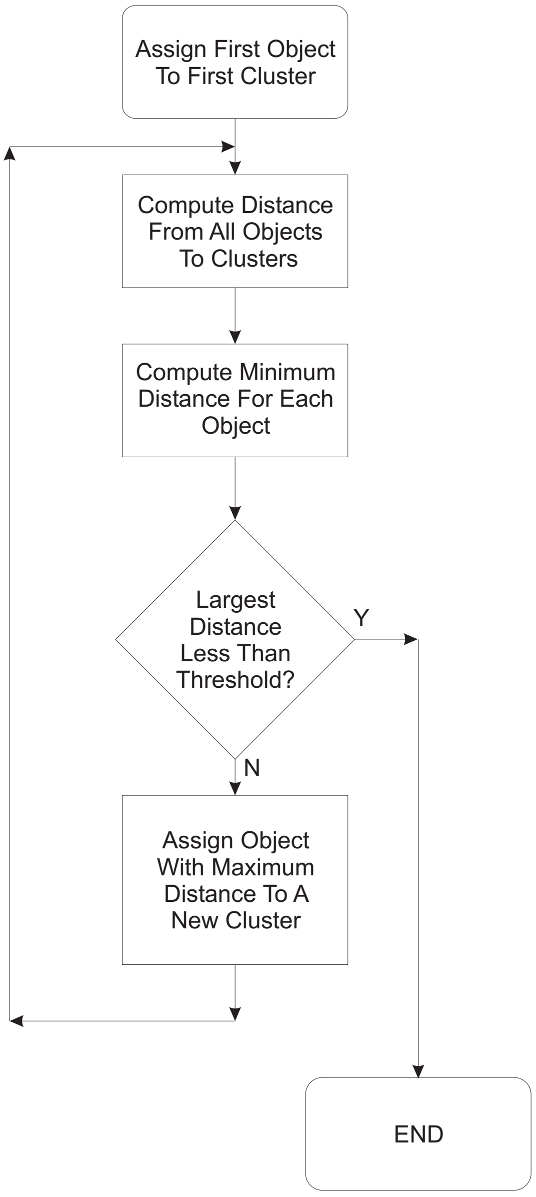
Assign First Object To First Cluster
Compute Distance From All Objects To Clusters
Compute Minimum Distance For Each Object
Largest Distance Less Than Threshold?
Y
N
Assign Object With Maximum Distance To A New Cluster
END

Figure 6.5: A flow diagram of the modified Batchelor and Wilkins' algorithm.

The Batchelor and Wilkins clustering method can have differing results depending on the order in which the data are presented to the clustering algorithm. To overcome this problem, the data were presented in a random order. This process was repeated fifteen times. This resulted in fifteen sets of clusters. The output that occurred the most was selected. This was used as the output from the clustering algorithm. This resulted in a considerable increase in the amount of time taken to cluster the data, but also resulted in more consistent results.

Chapter 7

Finding The Optimal Clustering Algorithm

As has been mentioned in the previous chapter, there are a large number of clustering methods that can be used. It is unlikely that all of them will perform equally well, so it is necessary to test the clustering algorithms on a number of data sets to see how well they perform. Once this has been done, it will be possible to select the best clustering algorithm for this task. It should be noted that although a clustering algorithm might outperform others for this data set it will not necessarily outperform other clustering algorithms on all data sets.

The testing of the individual clustering algorithms not only determined which algorithm performed best, but also determined the optimal settings for each algorithm's parameters.

7.1 Data set generation

In order to be able to test the different clustering algorithms, it is necessary to have a test data set that can be given to all of the clustering algorithms. The outputs of the algorithms can then be compared to see which one has performed better. It is also required that there be some benchmark against which the outputs of the clustering algorithms can be compared in order to determine how effective they are.

Since no data sets were available from external resources, it was necessary to generate some. The process for extracting the sets of *x* and *y* co-ordinates from the video input is the same as the one described in chapter 2. In total, fifteen data sets were generated. Each data set had various movements that were made, and had different durations.

Ideally the data set would have consisted of ‘long term data’. That is data that were collected over a number of days, observing normal behaviour. Unfortunately it was impractical to do this because of the large amounts of video data that would need to be stored. It would also be a monumental task to segment this data by hand. For these reasons subjects were asked to walk sets of patterns.

Different people were used in the generation of the data sets. This was done because people have differences in the way they walk. It also meant that there was a larger degree of freedom in how the task was interpreted.

The subjects were asked to walk various patterns around the office space. They were asked to go from one point to another, and then to stop there for approximately 3 or 4 seconds. They were then to walk somewhere else.

It was noted that some of the subjects did not pay particular attention to their start and endpoints of their movements, as well as not always remaining still for long enough (this helps the pre-processing stage identify individual movements. See chapter 2).

This process was used to created fifteen data sets that had between 14 and 60 movements each. Table 7.1 shows the details of the benchmark data set.

Data Set Number	Person Walking	Total Movements	Different Movements
1	Gordon	33	15
2	Gordon	33	11
3	Gordon	27	16
4	Gordon	34	4
5	Gordon	29	5
6	Gordon	32	7
7	Gordon	46	9
8	Gordon	34	10
9	Gordon	14	6
10	Gordon	26	10
11	Liza	44	14
12	Liza	45	10
13	Roshan	60	18
14	Roshan	37	16
15	Brett	46	21

Total Movements	540

Table 7.1: The data sets used for clustering.

7.2 Algorithms tested

Three clustering algorithms were tested against the benchmark output. The three algorithms used were:

- Modified Batchelor and Wilkins' Algorithm
- Graph Theoretic Clustering
- Agglomerative Hierarchical Clustering

Each of these algorithms was used to cluster the fifteen data sets into groups of similar movements. These groups (clusters) of movements were then compared

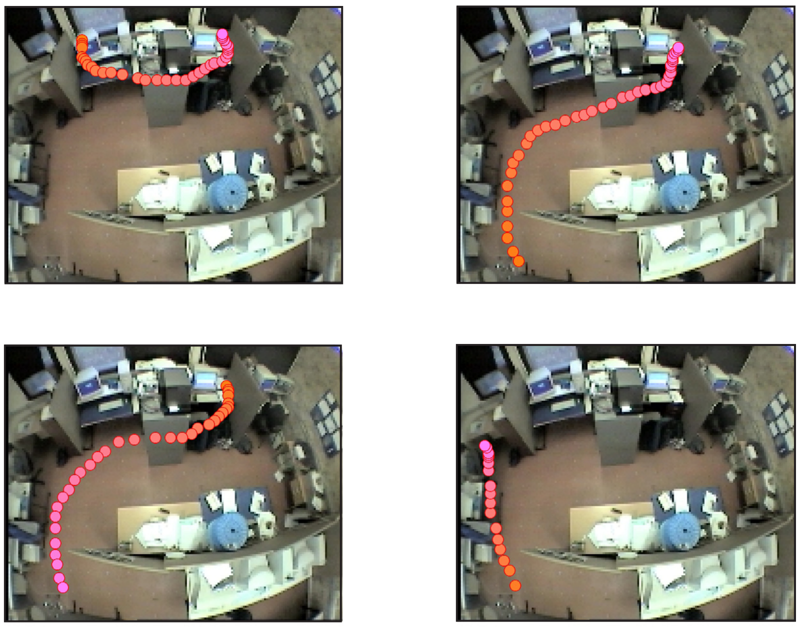

Figure 7.1: Some of the movements walked in the test data sets.

with those generated by hand (the benchmark data set). Since there are two parameters that can be adjusted for each of the algorithms, the clustering algorithms were each run with various parameter settings. These settings are shown in the table 7.2.

Clustering Algorithm	Relaxation of Endpoints	Similarity Level Threshold	Inconsistency Cuttoff
Graph Theoretic	0 to 19	5 to 295	n/a
Batchelor and Wilkins	0 to 19	5 to 295	n/a
Hierarchical	0 to 19	n/a	0.75 to 1.5

Table 7.2: Table of parameters for various clustering algorithms.

7.3 Comparing data sets

Comparing the output data sets generated by the clustering algorithm with those generated by hand is not as straight forward as conventional comparisons are. If data are to be clustered into various different categories, for instance A, B, C and D, it can easily be checked against the output of hand clustering. For example if the computer's output is:

Object number **11** is of type **A**

and the hand clustering output is:

Object number **11** is of type **B**

Then clearly object number 11 is incorrectly classified. The situation is more complex when dealing with sets. What should be done in the following scenario where the computer's output is:

Objects number { **1,3,4,5,11** } form set **A**

and the hand segmentation was:

Objects number { **2,3,4,5** } form set **A**?

Clearly it would not be optimal to mark the computer clustering generated set as completely incorrect, as it is a much better solution than if the clustering algorithm had provided the answer as:

Objects number { **1,6,7,9,11** } form set **A**

Based on this, it was decided that the error value would be determined by the number of incorrect objects in the set. This would include both objects missing from the correct data set as well as extra objects that were not present in the correct data set. This, however, leads to a very large error value as all of the incorrect data are incorporated into the error value twice, once for being in the wrong place and once for not being in the right place. The algorithm was therefore modified to only count each error once. The algorithm is discussed below.

7.3.1 Similarity matrix

Since the output from the clustering algorithms do not label the output sets, it was necessary to first determining which sets were the most similar. To do this a similarity matrix was used. Each element in the similarity matrix, s_{ij} is equal to the number of corresponding elements between the benchmark data set number i and the clustering algorithm's data set number j. For example if the benchmark data set consists of:

$$\begin{aligned} b_1 &= \{o_3, o_4, o_5, o_6, o_8\} \\ b_2 &= \{o_1, o_2\} \\ b_3 &= \{o_7, o_9, o_{10}\} \end{aligned} \tag{7.1}$$

and the clustering algorithm's data set consists of:

$$\begin{aligned} c_1 &= \{o_7, o_8, o_9, o_{10}\} \\ c_2 &= \{o_1, o_3\} \\ c_3 &= \{o_2, o_4, o_5, o_6\} \end{aligned} \tag{7.2}$$

Then the similarity matrix would be:

$$S = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \\ 3 & 0 & 0 \end{pmatrix} \end{array} \tag{7.3}$$

Once the similarity matrix has been calculated, each cluster in the benchmark data set is matched to a data set from the clustering algorithm. If there are fewer data sets from the clustering algorithm, then it is possible that a data set from the benchmark set will be matched with a null (empty) set. If there are more data sets in the clustering algorithm, then some of them will not be matched. The algorithm used is shown below.

Repeat:

1. Determine the maximum number of matches for each set in the benchmark data set.
2. Determine which set has the most matches.
3. Mark this benchmark data set and the clustering data set as similar sets.
4. Remove both of these sets from the similarity matrix.

until all the benchmark data sets have been removed, or until all of the clustering data sets have been removed. If the latter is the case, mark the unmarked benchmark sets as similar to the null (empty) set.

After this has been done, the total number of correct matches can be easily determined. The error value between the data sets is set as the number of incorrect matches.

7.3.2 An example

Based on the similarity matrix in equation 7.3:

1. The maximum values for the matches between the benchmark data set and the clustering data set is

$$\max(S) = \begin{pmatrix} 3 \\ 1 \\ 3 \end{pmatrix} \tag{7.4}$$

2. The maximum of these values is 3, which occurs twice. The first occurrence is arbitrarily chosen, so cluster b_1 is linked with cluster c_3.

3. These clusters are removed from the similarity matrix, leaving:

$$S = \begin{array}{c} \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix} \end{array} \tag{7.5}$$

4. The new maxima are calculated:

$$\max(S) = \begin{pmatrix} 1 \\ 3 \end{pmatrix} \tag{7.6}$$

5. The largest value is 3, so clusters b_3 is linked to c_1

6. This leaves b_2 to be linked with c_2

7. The results of determining the similar clusters are shown in the table below:

Benchmark Cluster	Benchmark Data	Computer Data	Computer Cluster
b_1	$\{o_3, o_4, o_5, o_6, o_8\}$	$\{o_2, o_4, o_5, o_6\}$	c_3
b_2	$\{o_1, o_2\}$	$\{o_1, o_3\}$	c_2
b_3	$\{o_7, o_9, o_{10}\}$	$\{o_7, o_8, o_9, o_{10}\}$	c_1

8. The objects that are present in the benchmark that are not present in the computer cluster are counted to determine the errors. This table should not be mistaken for an indication of the errors in the matching of the classes. For instance, b_3 is not a perfect match with c_1, although it might appear this way from the table.

Benchmark Cluster	Error Objects
b_1	o_3 , o_8
b_2	o_2
b_3	

9. There are a total number of 3 error objects, so these sets are given a similarity value of $7/10 = 70\%$.

7.4 Results

The graph theoretic, hierarchical and Batchelor and Wilkins' clustering methods were all applied to the fifteen data sets with various parameter settings. Optimal parameter settings were determined for each clustering algorithm.

7.4.1 Graph theoretic

As can be seen from the error surfaces in figure 7.2, it is important that the similarity threshold (match level) parameter is not too small. When this happens, very few patterns are deemed similar, resulting in a high error rate. It can also be seen that in some cases, when this parameter is too high, dissimilar patterns are matched together, again resulting in high error rates.

The 15 different error surfaces were merged to determine the effect of the parameters over all of the data sets. The resulting error surface is shown in figure 7.3.

From this error surface, the minimum value was located, and the appropriate parameter values were determined:

Similarity Measure (Match Level)	85
Endpoint Relaxation (DTW_K)	0

	Number of Movements	Incorrectly Classified	Correctly Classified	Percentage Correct
Data Set 1	33	1	32	97
Data Set 2	33	1	32	97
Data Set 3	27	1	26	96
Data Set 4	34	0	34	100
Data Set 5	29	0	29	100
Data Set 6	32	0	32	100
Data Set 7	46	0	46	100
Data Set 8	34	0	34	100
Data Set 9	14	0	14	100
Data Set 10	26	0	26	100
Data Set 11	44	2	42	95
Data Set 12	45	1	44	98
Data Set 13	60	7	53	88
Data Set 14	37	1	36	97
Data Set 15	46	7	39	85
Total	540	21	519	96

Table 7.3: Results from clustering using graph theoretic methods.

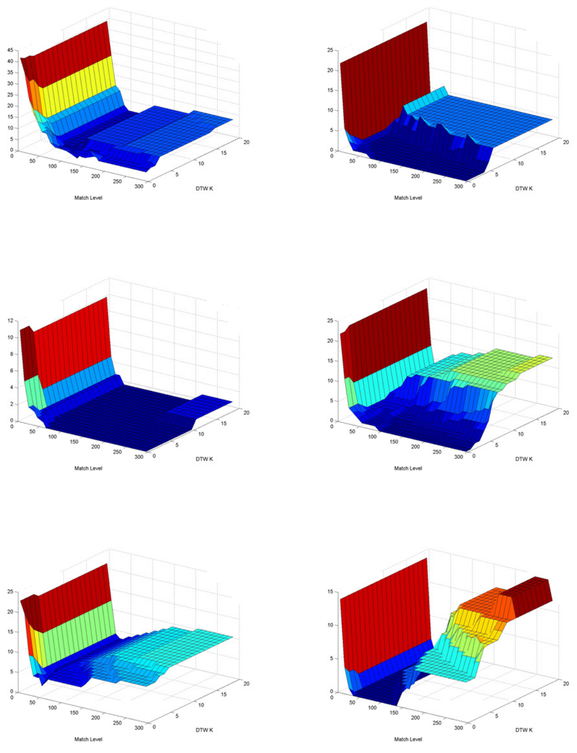

Figure 7.2: Various error surfaces for graph theoretic methods.

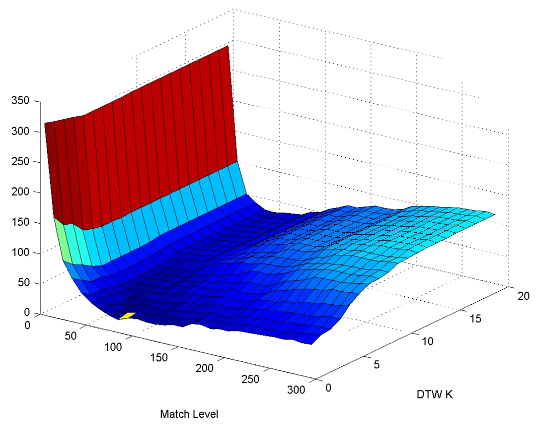

Figure 7.3: Total error surface for graph theoretic methods. The yellow square indicates the minimum.

7.4.2 Hierarchical clustering

The error surfaces for the hierarchical clustering technique vary quite considerably. However, one common feature is that the error rate seems to be low when the inconsistency cutoff parameter is in the order of 1 to 1.2. The total error surface (shown in figure 7.5) indicates that the optimal value for the inconsistency cutoff is 1.125.

Inconsistency Cutoff	1.125
Endpoint Relaxation (DTW_K)	5

The overall results of the hierarchical clustering are much worse than those of the graph theoretic methods. This is due to the nature of the clustering process which requires a very precise inconsistency cutoff parameter. This parameter varies from data set to data set. This means that it is not a feasible method for clustering this data as the parameter needs to be set before the clustering process begins. A table showing the results of the optimal parameter settings on the data set is shown below:

	Number of Movements	Incorrectly Classified	Correctly Classified	Percentage Correct
Data Set 1	33	11	22	67
Data Set 2	33	7	26	79
Data Set 3	27	10	17	63
Data Set 4	34	4	30	88
Data Set 5	29	4	25	86
Data Set 6	32	4	28	88
Data Set 7	46	11	35	76
Data Set 8	34	9	25	74
Data Set 9	14	3	11	79
Data Set 10	26	1	25	96
Data Set 11	44	17	27	61
Data Set 12	45	23	22	49
Data Set 13	60	16	44	73
Data Set 14	37	9	28	76
Data Set 15	46	16	30	67
Total	540	145	395	73

Table 7.4: Results from clustering using hierarchical clustering.

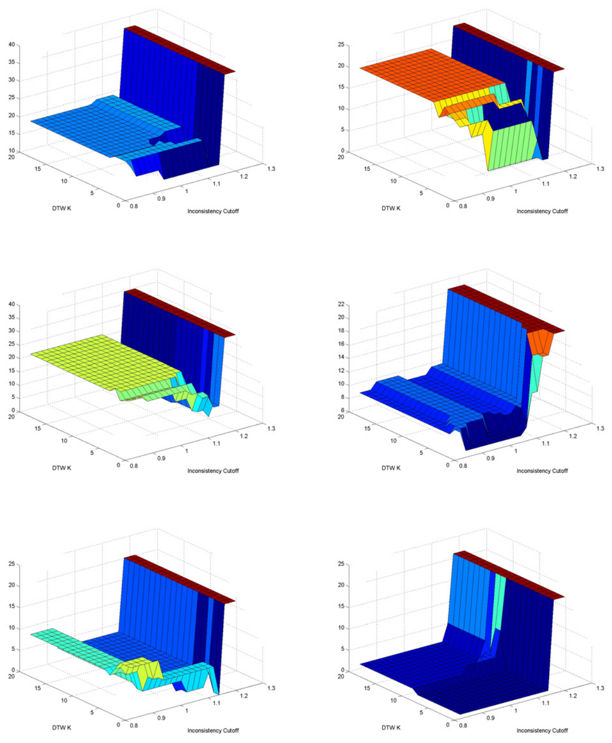

Figure 7.4: Various error surfaces for the hierarchical clustering method.

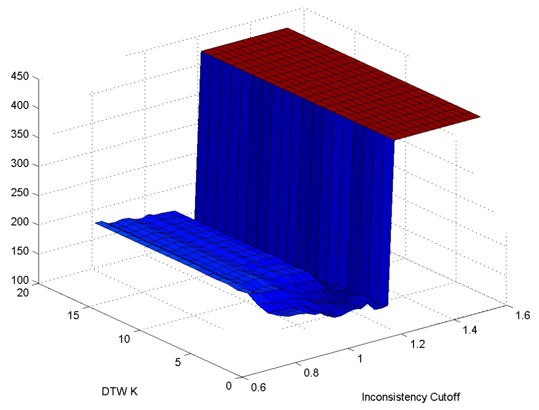

Figure 7.5: Total error surface for the hierarchical clustering method.

7.4.3 Batchelor and Wilkins

The error surfaces from the Batchelor and Wilkins maximum distance clustering algorithm are similar to those of the graph theoretic methods. However the minimal value occurs at a much larger match level value. This is because of step 3 in the graph theoretic method which effectively makes its true match level twice the specified match level value. This is why the Batchelor and Wilkins method has a match level value of approximately twice that of the graph theoretic method.

The optimal parameter settings found were:

Similarity Measure (Match Level)	155
Endpoint Relaxation (DTW_K)	2

A table showing the results of the optimal parameter settings on the data set is shown in table 7.5:

	Number of Movements	Incorrectly Classified	Correctly Classified	Percentage Correct
Data Set 1	33	0	33	100
Data Set 2	33	0	33	100
Data Set 3	27	2	25	93
Data Set 4	34	0	34	100
Data Set 5	29	0	29	100
Data Set 6	32	0	32	100
Data Set 7	46	0	46	100
Data Set 8	34	0	34	100
Data Set 9	14	0	14	100
Data Set 10	26	3	23	88
Data Set 11	44	3	41	93
Data Set 12	45	3	42	93
Data Set 13	60	7	53	88
Data Set 14	37	7	30	81
Data Set 15	46	5	41	80
Total	540	39	510	94

Table 7.5: Results using Batchelor and Wilkins’ clustering method.

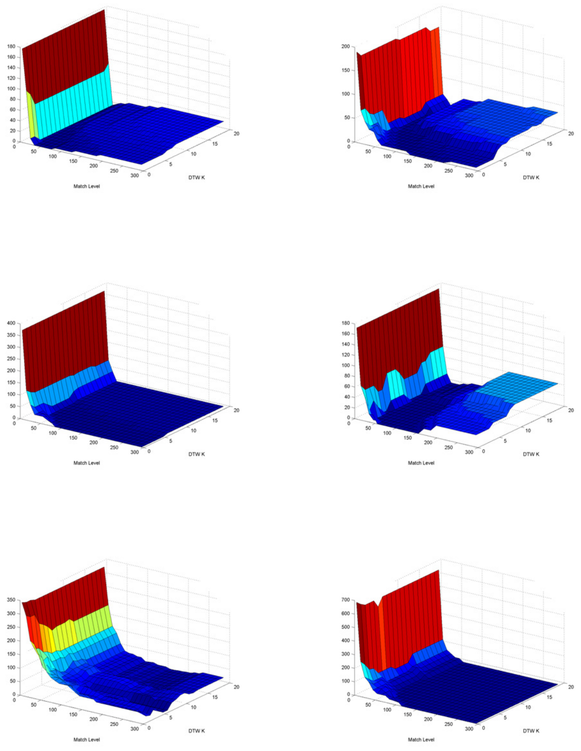

Figure 7.6: Various error surfaces for Batchelor and Wilkins' maximum distance clustering method.

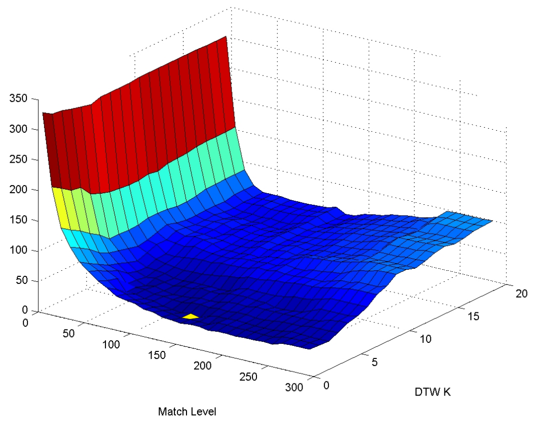

Figure 7.7: Total error surface for Batchelor and Wilkins' maximum distance clustering method. The yellow square indicates the minimum.

7.5 Optimisation

Two optimisations can be performed to increase the speed at which the clustering algorithms operate. Depending on certain parameter settings, they can affect the actual clustering output (though not necessarily in a negative way). They can also be implemented such that the clustering is not affected, the speed of computation being merely increased. A method of applying a variable distance measure to the warping path is also suggested.

7.5.1 Endpoint checking

One of the simplest checks that a human will use to decide on whether or not two movements are similar is to look at where the person being tracked started walking from and where they finished walking to. If these points are different in the two movements, then the movements will be considered different. If they are the same, then further analysis is needed. One must still decide whether the intermediate points are suitably similar. This technique can be easily implemented. It can also be processed very quickly and so a lot of time consuming DTW comparisons can be ‘tossed out’ by quickly determining that the patterns are dissimilar.

One simple method that achieves this is to compare the first and last six (this number is derived heuristically) data points of each movement. The one signal’s first six data points are slid across the other one’s. A subtraction is performed, the values are then squared and the lowest three numbers are summed and stored. After all possibilities have been checked, the lowest value of the stored numbers is used as a representative of the similarity between the two start points. A low value means they are very similar.

The same process is repeated for the endpoints. If both the similarity values for the start and the endpoint are beneath a certain threshold (also derived heuristically), then the DTW is used to determine a final similarity value. If one (or both) of them is above the threshold, no DTW is performed and they are given a very low similarity value.

7.5.2 DTW accumulator check

The DTW algorithm can be easily modified so that once the total accumulated path distortion increases above a critical value, the algorithm is stopped and the signals being compared are give an extremely low value for their similarity measure. This will cause a large increase in speed for cases where highly dissimilar signals are being matched.

The value to stop the DTW algorithm on can be set easily by taking the maximum desired similarity value and multiplying it by the normalisation factor of the DTW algorithm. However, this does not guarantee that all dissimilar signals will not be processed, but rather that all signals that are not processed to completion are dissimilar. It is expected that a lower value can be used while still getting extremely good results, but one cannot guarantee that a set of signals will not be discarded when they in fact are similar. If a lower value is used, it is important to perform a set of tests so that an appropriate value can be found.

7.5.3 A variable distance measure

A method that incorporates the idea of the endpoint checking technique described above could be used to further enhance the ability of the DTW algorithm to provide similarity measures between various movements. As has been mentioned before, for movements to be deemed similar, it is imperative that the start and endpoints are very similar. The intermediate points are not as important, although clearly they must still be within a reasonable distance of each other.

As can be seen in figure 7.8, the start and endpoints have a stronger weighting and so it is more important that they have a good match than the centre of the movement.

The implementation of such a system is not easy. The initial increase in weighting of the distance measure at the start point is simple enough to introduce, as the number of moves in the warping lattice from the start point is always known. However, the same is not necessarily true for the endpoint, as the optimal path has

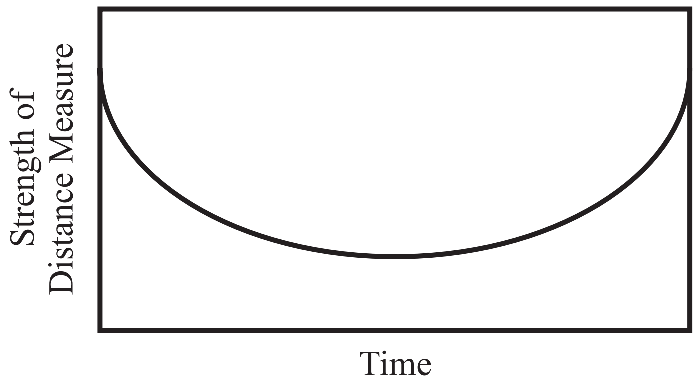

Figure 7.8: The proposed variable distance measure.

not yet been found. It is always possible to estimate the number of moves to the endpoint, but the accuracy of such an estimate has yet to be determined.

7.6 Conclusions

From the results, it can be clearly seen that the agglomerative hierarchical clustering algorithm does not perform as well as the other clustering techniques on this data set. This is due to the need of fine-tuning the inconsistency cut-off parameter for each data set. Since this parameter needs to be different for each data set it is impossible to find one value that will result in a good clusters for all data sets.

Both Batchelor and Wilkins' maximum distance clustering algorithm and the graph theoretic method produce good results. It should also be noted that the optimal parameters for both methods are similar.

Although the graph theoretic method outperformed the Batchelor and Wilkins clustering algorithm, some differences in their operation may still affect which algorithm should be used in a final implementation:

- The graph theoretic method uses all of the data present to train the system. This means that if more data are collected at a later stage, the whole system will need to be retrained. This could be time consuming.

- The Batchelor and Wilkins method is dependent on the order in which the data are presented to it. This means that different clusters could be generated for the same input data set. However, since the data are processed in a serial manner, all new data can be quickly trained without the requirement of retraining over the whole data set.

Chapter 8

Discovering Frequent Episodes

Most of this chapter is based on the work by Manilla and Toivonen [20, 21, 28] on the WINEPI algorithm.

8.1 Event sequences

Once the data stream of movement data has been analysed, and the individual movements clustered into similar ones, the original data stream can be transformed into an event sequence. This is in effect a highly compressed form of the data that was present in the initial data stream. In this work, each event corresponds to either an individual movement, or to a duration of no movement. The latter is not always used. In some cases it is more useful to ignore the times of no movement between the individual movements.

The event sequence created from the data stream in figure 8.1 is:

$$A\,B\,C\,B\,C\,B\,C\,B\,D\,E\,A\,F$$

This sequence does not contain any events for areas of no movement. A new event, Z, can be used to indicate areas of no movement. This results in a new

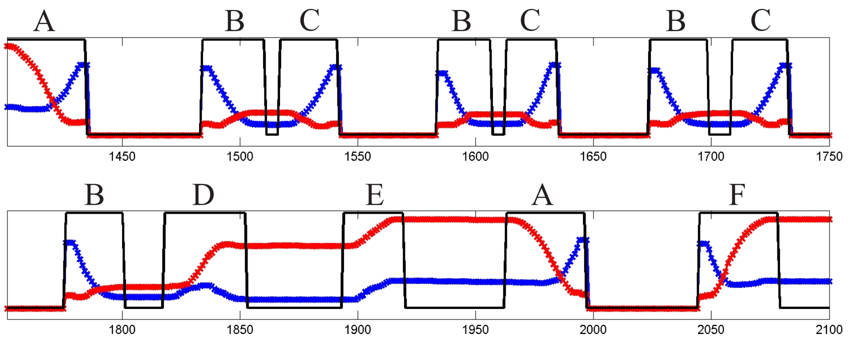

Figure 8.1: Transforming a data stream to an event sequence.

event sequence:

$$A\,Z\,B\,C\,Z\,B\,C\,Z\,B\,C\,Z\,B\,Z\,D\,Z\,E\,Z\,A\,Z\,F$$

It is necessary to have a user-specified threshold value for the amount of time that must pass without any movement occurring that will result in the inclusion of a Z event. This idea can be modified further to insert multiple Z events between movements depending on the duration of no movement (i.e. the longer the period of no movement, the more Z events that will be included into the event sequence.) An example of such a sequence is:

$$A\,Z\,Z\,B\,C\,Z\,Z\,B\,C\,Z\,Z\,B\,C\,Z\,Z\,B\,Z\,D\,Z\,Z\,E\,Z\,Z\,A\,Z\,Z\,F \tag{8.1}$$

A final method for obtaining an event sequence is to store the time at which each event occurred. This eliminates the need for the Z event, but does not lose any of the information with regard to the intervals of no movement between the individual events. An example is given below:

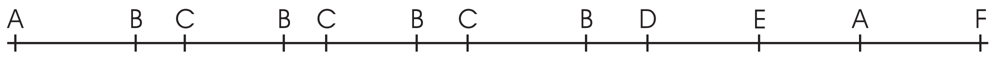

Figure 8.2: An alternative method for storing an event sequence.

It is necessary to decide whether the time marker for each event will be marked at its start point, midpoint, or endpoint. The choice of marking point could influence the output of further computation as similar events have different durations.

For the rest of this section, the event sequences dealt with will be of the same type as that shown in figure 8.2. However, it is a relatively simple task to transform the other methods of representing the event sequence to this style of representation.

8.1.1 Notations

The notations and formulae used in this section are based on those of Manilla *et al.*[20, 21, 28]. E is the set of *event types*. For the event sequence in figure 8.2,

$$E = \{A, B, C, D, E, F\} \tag{8.2}$$

Each *event* is a pair (A, t) where A is an *event type* $(A \in E)$ and t is the time at which the event occurred (this is usually given as an integer value, but the system can be transformed to deal with real values as well.)

An *event sequence*, **s**, is a triple (s, T_s, T_e), where

$$s = \langle (A_1, t_1), (A_2, t_2), ..., (A_n, t_n) \rangle \tag{8.3}$$

s is an ordered sequence such that $t_i \leq t_{i+1}$ for all $i = 1, .., n-1$. T_s is the starting time and T_e is the ending time (both integer values). $T_s \leq t_i \leq T_e$ for all $i = 1, ..., n$.

Figure 8.1 is the graphical representation of the event sequence $\mathbf{s} = (s, 1410, 2100)$ where

$$s = \langle (A, 1410), (B, 1483), (C, 1518), (B, 1590), ..., (F, 2046) \rangle \tag{8.4}$$

8.1.2 Windows

A *window* on an event sequence $\mathbf{s} = (s, T_s, T_e)$ is an event sequence $\mathbf{w} = (w, t_s, t_e)$ where $t_s < T_e$, $t_e > T_s$ and w is formed from the pairs (A, t) from s where $t_s \leq t < t_e$. The *width* of the window is the time span $t_e - t_s$. $\omega(\mathbf{s}, win)$ is the set of all windows on **s** such that $width(\mathbf{w}) = win$.

The first and last windows on a sequence extend outside the sequence. This ensures that an event will occur in the same number of windows irrespective of where it is located in the sequence.

8.2 Episodes

An episode is a partially ordered collection of events occurring together [20]. Generally they take the form of either serial or parallel episodes. However, more complex episodes can easily be generated by using combinations of serial and parallel episodes. Episodes can be easily described as directed acyclic graphs [20]. A few examples are shown below in figure 8.3.

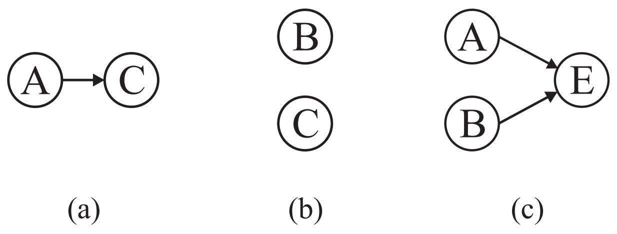

Figure 8.3: Graphic representation of various episodes [20].

Figure 8.3(a) shows a serial episode. Such an episode will occur when event *C* follows event *A*. Event *C* does not necessarily have to follow *A* immediately for the episode to occur, however if it is desired that event *C* follows immediately, this can be easily achieved by selecting the appropriate window ($width(\mathbf{w}) = 2$).

Figure 8.3(b) shows a parallel episode. For this episode to occur, both the *B* and *C* events must be present in the window. However, the order in which they occur is not important.

Figure 8.3(c) shows a more complicated episode that is neither serial nor parallel. For this episode to occur, both the *A* and *B* events must precede the *E* event. The order in which the *A* and *B* events occur is not important.

Serial episode detection is useful when the temporal order of the events is important. Since people will only make one movement at a time it seems likely that serial episode detection will be the most effective method for finding patterns in people's movements.

Parallel episode detection is suited to detecting when certain events happen at the same time or close together in time. This is suited for the analysis of alarm systems. For example it is useful to know that in a fire alarm system, a rapid increase in temperature and the smoke detector going off normally happen very soon after each other. But the order in which they occur might vary.

8.3 Finding frequent episodes

By searching through large volumes of data that store the various movements that a person has made over a long period of time, one can determine which episodes occur frequently. Once this has been done, this information can be used to predict future movements made by the person being tracked.

The *frequency* of an episode is defined as the fraction of windows in which that episode occurs. The frequency of episode α is:

$$fr(\alpha, \mathbf{s}, win) = \frac{|\{\mathbf{w} \in \omega(\mathbf{s}, win) \mid \alpha \text{ occurs in } \mathbf{w}\}|}{|\omega(\mathbf{s}, win)|} \tag{8.5}$$

We can now say that an episode is frequent if $fr(\alpha, \mathbf{s}, win) \geq \text{min\_fr}$. Where min_fr is a user-defined threshold.

This is particularly useful because if, for example, it is known that the serial episode $\{A,C\}$ occurs in 7% of the windows and that the serial episode $\{A,C,E\}$ occurs in 6% of the windows then it means that once the events $\{A,C\}$ have been detected in a window there is a $\frac{6}{7} = 85.7\%$ chance that event E will follow.

Unfortunately it is not as simple to generate such predictive rules from parallel episodes. This is due to them not containing any temporal information about the order of the events. For example if one knows that episode $\{A,B,C,D\}$ (window width of 4) occurs 20 times in an event sequence, and episode $\{A,B,C\}$ (window width of 3) occurs 25 times, it does not follow that episode $\{A,B,C,D\}$ will occur

80% of the time after episode $\{A,B,C\}$ has occurred. An example of this is shown below:

$$A\,B\,C\,E\,A\,B\,C\,E\,A\,C\,D\,B\,A\,D\,B\,C \tag{8.6}$$

Parallel episode $\{A,B,C\}$ is present twice in the event sequence. Parallel episode $\{A,B,C,D\}$ occurs three times, but never as a continuation of the $\{A,B,C\}$ episode. For this reason it is best to use parallel episodes to check for unusual behaviour only. This can be achieved by marking all episodes that occur above a user-specified threshold as frequent. Any episode that does not match one of the frequent episodes can be brought to security personnel's attention.

8.4 General algorithm

The algorithm for discovering which episodes are frequent is quite simple. It effectively loops through two main procedures:

- Candidate episode generation
- Candidate episode testing

The backbone of the search procedure is the lemma:

> *If an episode is frequent in an event sequence, then all sub-episodes are frequent as well.*

The algorithm starts by creating a list of candidate episodes. This is the entire alphabet of events. It then searches through the event sequence to determine which of these candidate episodes are frequent. A new set of candidate episodes is generated from the set of frequent candidate episodes. This process continues until the candidate episodes are the same size as the window or until no new candidate episodes are generated. An example of candidate generation for serial episodes is given below:

8.4.1 Candidate generation

Suppose that the following episodes have been found to be frequent in the event sequence:

$$\begin{gathered} A, B, C \\ A, B, G \\ A, B, H \\ B, C, H \\ C, D, E \\ F, F, C \end{gathered}$$

Episodes that have the same first `length-1` events must be merged together. Starting from the top:

$$A, B, C \text{ and } A, B, G \rightarrow A, B, C, G \tag{8.7}$$

The sub-episodes of length `length-1` must all be frequent for the candidate to be accepted. In this case, the sub-episodes are:

$$\begin{gathered} A, B, C \\ B, C, G \end{gathered}$$

However, the sub-episode B, C, G is not frequent. This means that the episode A, B, C, G *cannot* be frequent. Therefore it is discarded, and the next episodes are merged:

$$A, B, C \text{ and } A, B, H \rightarrow A, B, C, H \tag{8.8}$$

The sub-episodes are:

$$\begin{gathered} A, B, C \\ B, C, H \end{gathered}$$

Both of these sub-episodes are frequent, so A, B, C, H is added to the list of new candidate episodes.

This process is repeated until all possibilities of new candidate episodes have been exhausted.

The generation process for parallel and serial episodes is very similar. The only difference being that the parallel episode generation is quicker as it does not have to check as many possibilities as the serial case. For example, when merging A, B, C and A, B, E, the serial generation has two possible candidates: A, B, C, E and A, B, E, C. However, these two episodes mean exactly the same if describing a parallel episode, and so the latter case can be ignored.

8.5 Finding episodes in sequences

8.5.1 Serial algorithm

Frequent serial episodes are found by making use of finite state automata. Each state of a finite state machine will correspond to an event. The complete automata will accept the various candidate episodes that have already been generated.

The algorithm works by moving through the event sequence from the start, one event at a time. Each time the first event of an automaton enters the window, a new state machine is created. When an automaton for episode α reaches its accepting state, the starting time of the window is saved in the variable α`.inwindow`. When the automaton leaves the window, the variable α`.count` is incremented by the number of windows for which α was entirely in the window. This is only done if there are no other α automata in the accepting state.

8.5.2 Serial algorithm example

An example is given when looking for the candidate episodes:

$$\begin{aligned} \alpha &= A,B,D \\ \beta &= A,C,E \end{aligned}$$

in the event sequence:

$$s = A\,B\,A\,D\,C\,E \tag{8.9}$$

The window width is set to 4.

Each automaton column can contain one automaton of either type α or β. The subscript indicates the time at which the automaton started, and the brackets hold the events that have already happened, i.e. the state of the automaton. The st column indicates the start time at which the automaton reached its acceptance state (this does not necessarily always happen). α_{cnt} and β_{cnt} both hold the total number of occurrences of α and β. Note that this number only gets increased when the acceptance state is no longer present in the window. The horizontal lines indicate times at which automata get destroyed as their starting state is no longer present in the window. This is equivalent to when `Current_Time − Automaton_Start_Time = Window_Length`.

Time	Window	Automaton	st	Automaton	st	Automaton	st	Automaton	st	α_{cnt}	β_{cnt}
1	$w = \sqcup\sqcup\sqcup A$	$\alpha_1(A)$		$\beta_1(A)$							
2	$w = \sqcup\sqcup AB$	$\alpha_1(A,B)$		$\beta_1(A)$							
3	$w = \sqcup ABA$	$\alpha_1(A,B)$		$\beta_1(A)$		$\alpha_3(A)$		$\beta_3(A)$			
4	$w = ABAD$	$\alpha_1(A,B,D)$	4	$\beta_1(A)$		$\alpha_3(A)$		$\beta_3(A)$			
5	$w = BADC$	$\alpha_3(A)$		$\beta_3(A,C)$						1	
6	$w = ADCE$	$\alpha_3(A)$		$\beta_3(A,C,E)$	6					1	
7	$w = DCE\sqcup$									1	1
8	$w = CE\sqcup\sqcup$									1	1
9	$w = E\sqcup\sqcup\sqcup$									1	1

8.5.3 Parallel algorithm

Frequent parallel episodes are found by keeping track of how many events of the episode are currently in the window. When this counter is equal to the length of the episode, then the episode is in the window. Once again, the start time at which the entire episode is present in the window is noted. When the episode is not present in the window any more, the length of time it was present is calculated and added to the main counter.

8.5.4 Parallel algorithm example

An example is given when looking for the candidate parallel episodes (order of events does not matter):

$$\alpha = A, B, D$$
$$\beta = A, C, E$$

in the event sequence:

$$s = ABADCE \tag{8.10}$$

The window width is set to 4.

The columns α_{events} and β_{events} show which events from the associated candidate episode are currently in the window. To the right of each of these columns is an st column which indicates the start time when the entire episode was present in the window. α_{cnt} and β_{cnt} both hold the total number of occurrences of α and β. Note that this number only gets increased when the all of the events in the episode are no longer present in the window.

Time	Window	α_{events}	st	β_{events}	st	α_{cnt}	β_{cnt}
1	$w = \sqcup\sqcup\sqcup A$	A		A			
2	$w = \sqcup\sqcup AB$	AB		A			
3	$w = \sqcup ABA$	AB		A			
4	$w = ABAD$	ABD	4	A			
5	$w = BADC$	ABD	4	AC			
6	$w = ADCE$	AD		ACE	6	2	
7	$w = DCE\sqcup$	D		CE		2	1
8	$w = CE\sqcup\sqcup$			CE		2	1
9	$w = E\sqcup\sqcup\sqcup$			E		2	1

8.6 Complicated episodes

Complicated episodes (those that are neither serial nor parallel) are more difficult to search for. They are not as easy to define as serial and parallel episodes because they encompass such a large variety of possibilities. However, some of these more complicated episodes can be described as a collection of serial and parallel episodes combined. For example, the complicated episode shown in figure 8.3 can be decomposed. This is shown in figure 8.4.

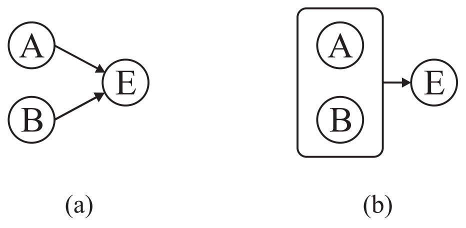

Figure 8.4: Graphic representation of the decomposition of a complex episode [20]; (a) original episode; (b) decomposed episode.

It now consists of a serial episode that has a parallel episode embedded within it. A practical way of searching for the presence of such complicated episodes is to first check whether or not all of the events are in the window. When they are all present, the ordering within the window can be checked.

Chapter 9

Finding Episodes in Real Data

The methods described in chapter 8 were applied to various sets of data to demonstrate their effectiveness at finding patterns in large sets of data. Unfortunately, only *artificial* data sets (data sets that were generated by specifically walking set patterns, rather than data sets generated by analysing usual behaviour over a long period of time) were present to test the algorithms on. There is no reason to expect poor performance on less artificial data sets. This is because the WINEPI algorithm guarantees to find *all* episodes in an event sequence.

Three different tests were performed to demonstrate the effectiveness of the WINEPI algorithm at finding episodes in data streams.

9.1 Test 1

The sequence of test data used in this test contained a person walking two set patterns. The first pattern involved four movements and was repeated five times. The second pattern consisted of three movements, and was repeated three times. It is important to note that the data used were the movements themselves, and not the start and endpoints of the movements. The start and endpoints are analysed in test 2. The sequence of movements that was analysed is shown below:

$$C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ E\ A\ B\ E\ A\ B\ E\ A\ B$$

This sequence was analysed to determine what frequent serial episodes existed. Window sizes varied from a width of two to six units. The results of the test are shown in tables 9.3 and 9.4. The rules generated in table 9.4 show only a selection of the rules that can be generated from the output (the rest can be found on the attached CD-ROM). It is a fairly simple task to automate the process to create various sets of rules. For example, all rules that have a confidence level of 75% or greater might be required. It is also possible to determine all rules concerned with a certain event or all of the rules that start or end with certain events. All of the rules generated can be used for the prediction of upcoming movements based on where the person being tracked has been. All rules that are not present (i.e. have a 0% chance of occurring) can be considered as unusual behaviour. If one of these movements takes place, an alarm can be activated to alert security personnel that an unusual activity is occurring.

9.2 Test 2

Test 2 used the sequence of locations of the person being tracked (whilst stationary) as the event sequence to search for frequent serial episodes. The same movement data is used as in test 1. The new set of events, and their corresponding locations in figure 9.1 is shown in table 9.2.

The event sequence for test 2 is:

$$A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ B\ C\ D\ A\ C\ D\ A\ C\ D\ A\ C\ D\ A$$

Once again, the sequence was analysed to determine what frequent serial episodes existed. Window sizes varied from a width of two to six units. The results of the test are shown in tables 9.5 and 9.6.

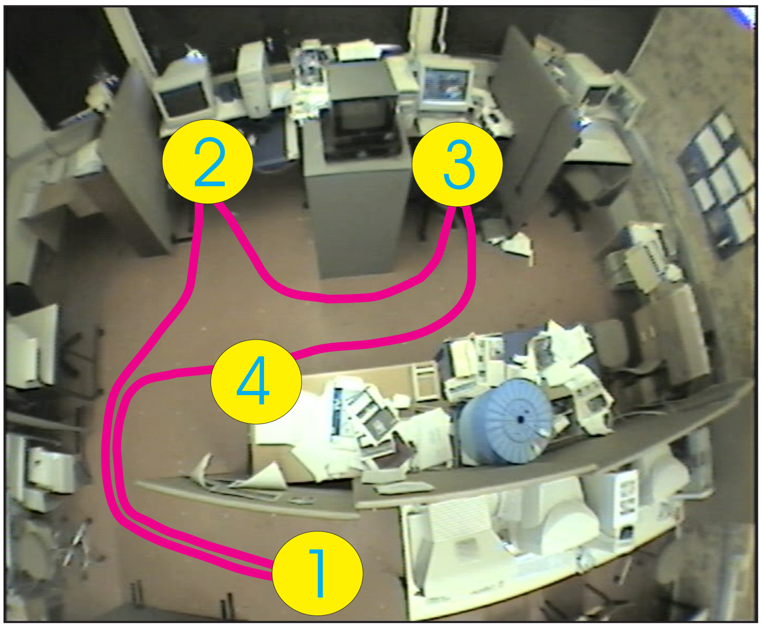

Figure 9.1: Start and endpoints for the various movements made in tests 1 and 2.

Movement	Start Point	Endpoint
A	3	4
B	4	1
C	1	2
D	2	3
E	1	3

Table 9.1: Movement labels and their corresponding start and endpoints as used in test 1.

Event	Location
A	1
B	2
C	3
D	4

Table 9.2: Locations and their corresponding events.

9.3 Test 3

Test 3 was designed to see how effectively the WINEPI algorithm could find a set of fixed rules. At each of the locations 1,2,3 and 4, a set of rules was placed that told the person being tracked where to move next, depending on the role of a die. This meant that the person being tracked did not decide where they would walk next. It was hoped that the algorithm would be able to find rules very similar to those implemented. The rules are shown in table 9.7.

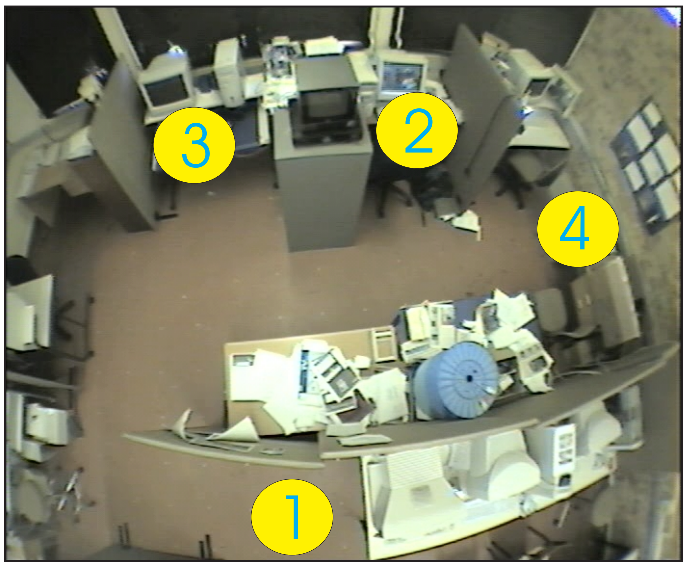

Figure 9.2: The four locations used in test 3.

Once again, the outputs from the segmentation and clustering algorithms were a set of movements rather than the set of locations visited. The table below shows

Episode	No. Occurrences	Episode	No. Occurrences	Episode	No. Occurrences
A	8	AB	8	ABC	4
B	8	BC	4	ABE	3
C	5	BE	3	BCD	4
D	5	CD	5	BEA	3
E	3	DA	5	CDA	5
		EA	3	DAB	5
				EAB	3
ABCD	4	ABCDA	4	ABCDAB	4
ABEA	3	ABEAB	3	ABEABE	2
BCDA	4	BCDAB	4	BCDABC	3
BEAB	3	BEABE	2	BCDABE	1
CDAB	5	CDABC	4	BEABEA	2
DABC	4	CDABE	1	CDABCD	4
DABE	1	DABCD	4	CDABEA	1
EABE	2	DABEA	1	DABCDA	4
		EABEA	2	DABEAB	1
				EABEAB	2

Table 9.3: Results from test 1: frequent serial episodes.

No.	Rule	%
1	$ABC \rightarrow DAB$	100
2	$DAB \rightarrow CDA$	80
3	$C \rightarrow DAB$	100
4	$C \rightarrow DABC$	80
5	$C \rightarrow DABE$	20
6	$D \rightarrow ABCDA$	80
7	$D \rightarrow ABEAB$	20

Table 9.4: Some of the rules generated as output from test 1. The “ →” means “ is followed by”

Episode	No. Occurrences	Episode	No. Occurrences	Episode	No. Occurrences
A	9	AB	5	ABC	5
B	5	AC	3	ACD	3
C	8	BC	5	BCD	5
D	8	CD	8	CDA	8
		DA	8	DAB	4
				DAC	3
ABCD	5	ABCDA	5	ABCDAB	4
ACDA	3	ACDAC	2	ABCDAC	1
BCDA	5	BCDAB	4	ACDACD	2
CDAB	4	BCDAC	1	BCDABC	4
CDAC	3	CDABC	4	BCDACD	1
DABC	4	CDACD	3	CDABCD	4
DACD	3	DABCD	4	CDACDA	3
		DACDA	3	DABCDA	4
				DACDAC	2

Table 9.5: Results from test 2: frequent serial episodes.

No.	Rule	%
1	$A \rightarrow B$	55
2	$AB \rightarrow CDA$	100
4	$C \rightarrow DABCD$	50
5	$C \rightarrow DACDA$	38
6	$D \rightarrow A$	100
7	$D \rightarrow ABCDA$	50
8	$D \rightarrow ACDAC$	25

Table 9.6: Some of the rules generated as output from test 2. The “ $\rightarrow$” means “ is followed by”

Current Location	Die Roll	New Location		Current Location	Die Roll	New Location
1	1,2,3	2		4	1,2,3	2
1	4,5,6	3		4	4,5,6	3
2	1,3,5	4		3	1,4,6	2
2	2,3	3		3	2,3	4
2	6	1		3	5	1

Table 9.7: Rules based on die throws for test 3.

the different movements detected, and how they relate to the locations show in figure 9.2

An event sequence was generated by walking to locations, rolling a die, and walking to the new location specified by the rules. The event sequence generated was:

B I L I K E H F L H E I L H F L G A E H F K F K F L H
F K F K E I L G A F K F L H E G B I K E H F L I K D

Due to the large number of events in this test, the list of results is very large. The full results can be found on the attached CD-ROM. A single case is considered, which demonstrates the effectiveness of the algorithm at finding patterns.

The results were analysed to see whether the rules for what to do when leaving location 2 had been detected. To achieve this, the events that have location 2 as their endpoint needed to be analysed. These events are A, H, K.

Table 9.9 shows the various episodes that resulted in the person being tracked arriving at location 2. It also contains the various locations that the person went to after arriving there. From the table it can be seen that when leaving location 2, movement D is performed $\frac{1}{17} = 6\%$ of the time, movement E is performed $\frac{6}{17} = 35\%$ of the time and movement F is performed $\frac{10}{17} = 59\%$ of the time.

The results shown in table 9.10 show that the rules 'learnt' are similar to those created. They show that the patterns as defined by the rules have been detected. There is some dissimilarity in the chance of the rule being executed. This is due

Movement	Start Point	Endpoint
A	1	2
B	1	3
C *	1	4
D	2	1
E	2	3
F	2	4
G	3	1
H	3	2
I	3	4
J *	4	1
K	4	2
L	4	3

Table 9.8: Movement labels and their corresponding start and endpoints for test 3.
*Impossible to appear in the event sequence (see table 9.7).

to the small period of time for which test data were generated. If a larger event sequence were present, the percentages would be closer to the desired values.

9.4 Discussion

The WINEPI algorithm of Manilla and Toivonen [20, 21, 28] performs very well at finding frequent occurrences of serial and parallel episodes in large data streams. There are a number of considerations that must be taken into account when trying to automatically determine frequent patterns in a data set.

9.4.1 Domain knowledge

As with many machine learning problems, the system cannot be expected to learn effectively without making use of as much knowledge about the work domain as

Episode	No. Occurrences		Episode	No. Occurrences		Episode	No. Occurrences
A	2		H	7		K	8
AE	1		HE	2		KD	1
AF	1		HF	5		KE	3
						KF	4

Table 9.9: Results from test 3: frequent serial episodes.

Movement	Discovered	Desired
D	6%	17%
E	35%	33%
F	59%	50%

Table 9.10: Results from test 3, showing the probability of certain movements happening compared to the probability as specified by the rules (see table 9.7.)

is possible. By applying this knowledge to the system, the task becomes more effectively constrained, and therefore the chances of success are increased. This knowledge can come in a variety of forms. For example, if it is known that there will be no correlation between events that happen more than 10 minutes apart from each other, then the window size can be effectively set. If it is known what form the patterns should take on, this information can be used to design a suitable template based on the serial and parallel episodes.

9.4.2 Serial or parallel episodes

Should the search be for serial episodes, parallel episodes or both? This question is related to the amount of knowledge that is present about the domain that is being worked in. In the case of searching for patterns in people's movements, both serial and parallel episodes are useful. Serial episodes are the most intuitive and will provide rules such as:

> *When the person goes to his desk, there is a* 45% *chance that his next movement will be to go and make coffee.*

Parallel episodes are also useful as they describe movements that will often happen in close proximity to each other, but not necessarily in a specific order. Serial episodes can be considered as being a special case of parallel episode.

9.4.3 Location data or movement data

As was shown in tests 1–3, the rules generated vary quite substantially depending on whether movement data or location data is used. Intuitively, location data seems to be a better choice, as it will provide rules such as where people move to after visiting certain locations. However, it does not contain any information on the movement between the two locations. For instance, if the person being tracked walked via a different route. This information is present in the movement data. The movement data also contains the location data, although it needs to be extracted from the data. This is not a problem if the final system learns the frequent patterns and make predictions on what is most likely to happen next. It is only a matter of concern if the desired output is a set of rules extracted from the process. In this case it might be better to make use of both data sets, although this will take considerably more processing time.

9.4.4 Data sets

When using a system to learn certain patterns in an environment, it is necessary to ensure that there is sufficient training data in the data set. As mentioned earlier in the results of test 3, the more training data that is available, the better. It is also necessary that the training data contain numerous occurrences of *all* of the cases that are to be detected. This can easily be monitored by placing a threshold value on the number of occurrences that must take place before rule generation is performed.

9.4.5 Event sequence representation

The event sequence representation (as discussed in section 8.1) can also affect the outcome of the episode detection process. If the Z events are used, then the event sequence will increase in length dramatically, especially if there are large time intervals between movements. This means that the window size will need to be extremely large to detect any useful patterns.

9.4.6 Multiple event sequences

The addition of an extra event sequence that contains different kinds of events such as time markers: 7:00 am, lunch time, etc. could be analysed with the movement event sequence. This could be used in conjunction with a search for parallel episodes to result in many useful rules such as:

The person goes to his desk soon after 7:00 am.

9.4.7 FPGAs

Lipson and Hazelhurst [18] have shown that FPGAs (field programmable gate arrays) can be effectively used for the fast detection of patterns in DNA sequences. It is highly possible that such techniques could be useful for increasing the speed of searches for specific patterns in data sets that are used in the WINEPI algorithm. The reconfigurability of FPGAs means that they could be used to search for all different patterns required. However, the time required to set up the FPGA could result in FPGAs only being useful in systems with extremely large data sets. This means that the additional time overheads for configuring the FPGA would need to be minimal compared to the overall time savings.

Chapter 10

Conclusions

10.1 Segmentation

A system has been developed that can effectively segment out a marker in a specifically constrained environment. This is particularly useful for generating large quantities of data that indicate a person's location over a long period of time. It is important to note that this segmentation algorithm has been designed for a highly constrained environment in which it performs well (real-time processing up to 15 frames per second). The segmentation system will not perform well under other conditions. A more robust segmentation algorithm that can accurately determine the locations of numerous people should be developed. Suitable post-processing functions that take account some of the short falls of the segmentation algorithm have been implemented to ensure a reliable data stream as output from the segmentation module.

10.2 Clustering

The DTW has been shown to be an extremely useful distance measure for determining the similarity between signals of differing lengths. It outperforms other techniques such as linear time normalisation and the euclidean distance measure. It is important to note that there are numerous constraints and optimisations that can be performed on the DTW in order to achieve better results. Certain optimisations have been performed to ensure that the DTW is best suited for matching the various movements made by people.

Three clustering techniques were tested to see how effectively they could perform unsupervised classification on fifteen data sets of people's movements. Each data set contained between 14 and 60 movements. Hierarchical clustering performed the poorest of the three clustering algorithms, achieving only 73% correct classification. The Batchelor and Wilkins algorithm achieved 94% correct classification and the graph theoretic method 96% correct classification.

It should be noted that this does not necessarily mean that the graph theoretic clustering method should always be used instead of Batchelor and Wilkins' clustering method since both algorithms have different advantages and disadvantages.

10.3 Finding frequent episodes

The WINEPI algorithm can effectively find both serial and parallel episodes in event sequences. This can be put to great effect in a security system as it can be used to predict people's movements. When the predictions are incorrect, the security personnel can be alerted to the fact that an something unusual has happened.

The downside of this algorithm is that it can take a very long time to process large data sets. However, this need not be a big constraint as the most likely final implementation of a system will use the WINEPI algorithm in a set training phase. This means that the algorithm can be run to completion in an offline mode. It does not have to deal with all of the subsequent data that will be present when the system is running online.

10.4 Implementation

This feasibility study has shown that it is possible to automatically detect the various patterns that are present in people's movements. It has also shown that it is possible to make predictions based on these movements. If the methods developed here are to be used in an online system, it is important that further investigations into the specific requirements of the system are performed. This is required so that the optimal parameter settings can be found.

10.5 Further Investigation

The obvious continuation of this feasibility study is to attempt to use the methods developed here for a system that makes use of location data that has been acquired from a normal work environment. This will pose numerous new problems to deal with, including:

- the real-time segmentation of multiple people from multiple cameras. This is an area of ongoing research and should be considered a separate project.
- the determining of the optimal parameters for the various algorithms presented. These include the various DTW constraints, clustering algorithm parameters, and window widths for the data mining algorithms. It is expected that there will not be one optimal set of parameters, but rather that there will be optimal parameter settings for different operating environments.
- the porting of some of the algorithms developed from Matlab to another language such as C++, in order to increase their execution times.

Other areas of further investigation include the simultaneous use of data streams from other algorithms to enhance the data mining algorithm's output. For example if the location data, is used in conjunction with a datastream indicating who is being tracked, and a data stream indicating the time of day, more meaningful rules might be extracted.

It would also be interesting to see how well these algorithms work on different data stream types (e.g. instrumentation and other signal processing outputs). The data streams would need to be similar to the walking data in that there are no large discontinuities in the data (unless the discontinuities are used for segmentation purposes).

Appendix A

Attached CD-ROM

The following files can be found on the attached CD-ROM:

A.1 Raw data

This file contains the raw data that were created by the pink hat segmentation algorithm. The raw data for the fifteen data sets used are shown.

A.2 Individual movements

This file contains the output from the pre-processing stage and the movement detection algorithm. The data for the fifteen data sets used are shown.

A.3 Clustering

This file contains the various outputs from the clustering techniques used on the fifteen data sets. The benchmark data set is also present.

A.4 Event sequences

This file contains the various event sequences that were generated from the fifteen data sets after graph theoretical clustering had been performed.

A.5 Serial episodes

This file contains the various serial episodes that were detected from the fifteen test event sequences.

A.6 Parallel episodes

This file contains the various parallel episodes that were detected from the fifteen test event sequences.

A.7 Serial rules

This file contains the serial rules that were generated from the fifteen test event sequences. Only rules which have a percentage certainty greater than or equal to fifty percent are shown. Furthermore, rules are only shown if they have occurred at least twice in the event sequence.

A.8 Test results

This file contains the results from the three tests performed. Only rules which have a percentage certainty greater than or equal to fifty percent are shown. Furthermore, rules are only shown if they have occurred at least twice in the event sequence.

Bibliography

[1] S Anand and J Hughes. Data Mining: Looking Past the Tip of the Iceberg. Lecture Notes, 2001. `http://divcom.otago.ac.nz/infosci/Courses/INFO233/2001/Lectures/`.

[2] Ascom. SEDOR - selflearning event detection in video surveilance. `http://www.sedor.ch`.

[3] S. Bow. *Pattern Recognition and Image Preprocessing*, chapter 5. Marcel Dekker Inc., 1992.

[4] J. C. Chappelier and A. Grumbach. A Kohonen map for temporal sequences. In *Neural Networks and Their Applications. Conference Proceedings*, pages 104–10. Domaine Univ. Saint-Jerome, Marseille, France, 1996.

[5] Leon Cooper and Mary Cooper. *Introduction to Dynamic Programming*. Pergamon Press, 1981.

[6] R Duda and P Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, Pittsburgh, PA, 1973.

[7] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[8] Julian Faraway. Cluster analysis. Lecture Notes for Statistics 503. `http://www.stat.lsa.umich.edu/˜faraway/stat503/lecnotes/`.

[9] N Ikizler. Mining sequential patterns: an overview. Term Paper. `http://www.cs.bilkent.edu.tr/˜inazli/`.

[10] The Mathworks Inc. *Statistics Toolbox User's Guide version 3*. The Mathworks Inc., November 2000. `http://www.mathworks.com/access/helpdesk/help/pdf_doc/signal/signal_tb.pdf`.

[11] A Jain, M Murty, and P Flynn. Data clustering: A review. In *ACM Computing Surveys*, volume 31, pages 264–323, September 1999.

[12] E. Keogh and M. Pazzani. An indexing scheme for similarity search in large time series databases. In *The 11th International Conference on Scientific and Statistical Database Management*, Cleveland, Ohio, 1999.

[13] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, 2001.

[14] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Knowledge Discovery and Data Mining*, pages 239–243, 1998.

[15] Eamonn J. Keogh and Michael J. Pazzani. Scaling up dynamic time warping to massive datasets. In *Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

[16] Eamonn J. Keogh and Padhraic Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Knowledge Discovery and Data Mining*, pages 24–30, 1997.

[17] T Lin. PAKDD2002 Workshop - Toward the Foundation of Data Mining. `http://www.mathcs.sjsu.edu/faculty/tylin/pakdd_workshop.html`.

[18] A Lipson and S Hazelhurst. DNA pattern matching using FPGAs. In *Proceedings of the twelth annual symposium of the Pattern Recognition Association of South Africa*, pages 180 – 185, November 2001.

[19] H Manilla, H Toivonen, and A Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proceeings of the First International Conference on Knowledge Discovery and Data Mining*, pages 210–215, August 1995.

[20] H Manilla, H Toivonen, and A Inkeri Verkano. Discovery of frequent episodes in event sequences. Technical Report C-1997-15, University of Helsinki, February 1997.

[21] H Mannila, H Toivonen, and A Inkeri Verkamo. Discovering frequent episodes in series. In *First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210 – 215. AAAI Press, August 1995.

[22] C. Myers, L. R. Rabiner, and A. E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-28(6):622, 1980.

[23] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*, pages 201–241. Prentice-Hall Inc., 1993.

[24] Lawrence R. Rabiner, Aaron E. Rosenberg, and Stephen E. Levinson. Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-26(6):575–582, 1978.

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

[26] Kerby Shedden. Cluster analysis (unsupervised learning) lecture notes for statitics 710. `http://www.stat.lsa.umich.edu/~kshedden/Courses/Stat710/lecture-cluster1.pdf`.

[27] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int'l Conf. on Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[28] Hannu Toivonen. *Discovery of frequent patterns in large database collections*. PhD thesis, University of Helsinki, 1996.

[29] Brendt Wohlberg. Investigations in sequence matching techniques applied to on-line signature verification. Master's thesis, University of Cape Town, 1993.

[30] Stuart N Wrigley. Speech recognition by dynamic time warping. http://www.dcs.shef.ac.uk/~stu/com326/.

[31] M J Zaki, S Parthasarathy, Mogihara, and W Li. New algorithms for fast discovery of association rules. In *In 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–296. AAAI Press, 1997.

[32] Mohammed Javeed Zaki. Efficient enumeration of frequent sequences. In *International Conference on Information and Knowledge Management*. ACM, SIGIR, and SIGMIS, 1998.

[33] Mohammed Javeed Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42:31–60, 2001.